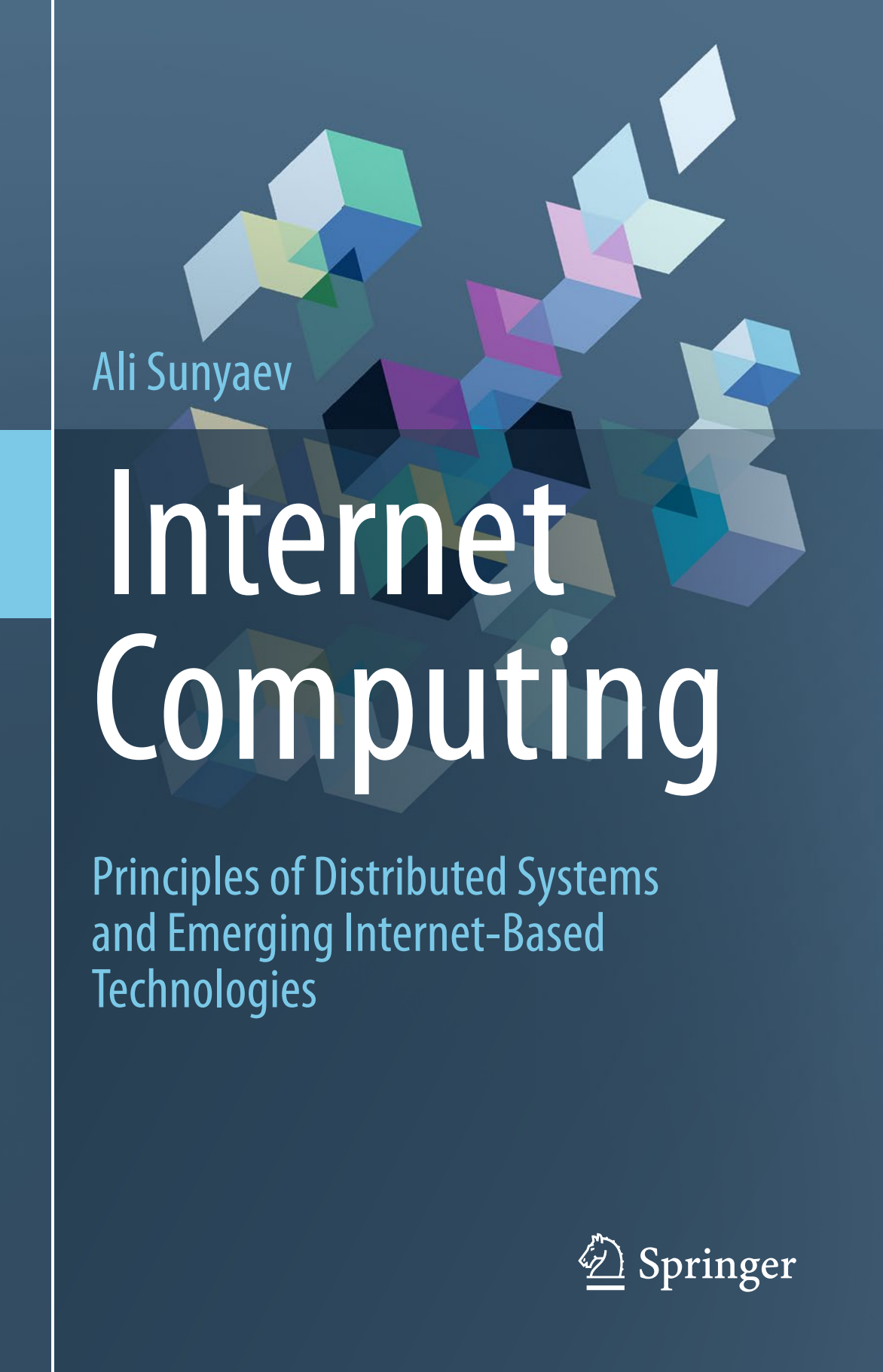




Ali Sunyaev

Internet Computing

Principles of Distributed Systems
and Emerging Internet-Based
Technologies



Springer

Internet Computing

Ali Sunyaev

Internet Computing

Principles of Distributed Systems and
Emerging Internet-Based Technologies

 Springer

Ali Sunyaev
Institute of Applied Informatics
and Formal Description Methods
Karlsruhe Institute of Technology
Karlsruhe, Germany

ISBN 978-3-030-34956-1 ISBN 978-3-030-34957-8 (eBook)
<https://doi.org/10.1007/978-3-030-34957-8>

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The Internet is a success story. Over the past few decades, its value to individuals, organizations, and society has become more profound than most of us could ever have imagined. This is no surprise, given that the Internet has brought fundamental changes to almost all areas of life. It has not only changed the way we communicate, gather information, and consume media, but has also led to profound societal changes. Important social debates (e.g., #MeToo) are currently held online, because the Internet provides the required infrastructure and communication channels to allow almost all of our society members to democratically exchange their viewpoints. The Internet's widespread dissemination has also led to new types of companies, purely based on Internet-related business models, emerging. Amazon, Google, and Facebook are famous examples of such companies.

The Internet landscape is still evolving steadily and becoming increasingly complex. This complexity is not only due to the emergence of its underlying infrastructure but also relates to organizations and individuals' more widespread use of the Internet's new and innovative tools. It is therefore extremely important that students and young professionals in the fields related to information technology familiarize themselves with the Internet's basic mechanisms and are introduced to the most promising Internet-related technologies of our time.

Against this backdrop, this book seeks to provide insights into the most important technologies and concepts related to the Internet through the scientific field of Internet computing. It is obviously not possible to cover all facets of Internet computing in a single book; instead, this book's goal is to provide a broad overview of the most important foundational concepts and to shed light on the most promising current trends in Internet computing. In doing so, I primarily take an organizational view and reflect on the importance of the constantly evolving Internet-related architectures and technologies for business issues. For example, the use of cloud-based services has, for organizations of all kinds, become widespread over the last decade. However, with an increasing number of organizations moving toward Industry 4.0 and the Internet of Things, the cloud paradigm may no longer adequately meet some of the emerging demands. Consequently, new paradigms, such as

Fog and Edge Computing, are coming into play and receiving increasing attention from research and practice. Nevertheless, other revolutionary technologies are also closely linked to the Internet and also have the potential to continue the fundamental change driven by the Internet. At the time of writing this book, distributed ledger technologies (e.g., the Bitcoin Blockchain) are just one of such hype topics. Many people presume that distributed ledger technologies' elimination of superfluous third parties could have the potential to lead to profound changes in many areas, such as finance, health, and politics. Whether and to what extent such changes will actually occur remain to be seen. Nevertheless, I believe that such promising developments should be covered and their basics conveyed to the readers of a book such as this.

The idea for this book was raised during discussions with my research associates and PhD students at Karlsruhe Institute of Technology (KIT). While searching for a well-known and respected textbook on Internet computing for one of our lectures, we were surprised to find that no such work existed. Consequently, I decided to write it. This book is divided into 12 chapters, which, as a whole, offer a comprehensive lecture on the foundations of Internet computing. Each individual chapter can be a lecture unit.

My sincerest thanks to all my research associates and PhD students at KIT who helped me create the texts: Tobias Dehling, Malte Greulich, Niclas Kannengießer, Theresa Kromat, Sebastian Lins, Benjamin Sturm, Heiner Teigeler, and Scott Thiebes. I specifically want to thank Manuel Schmidt-Kraepelin, who not only helped me create the book's content but also supported me during the publication process. I would also like to thank Elisabeth Lieder for supporting the project at the operational level and Deniz Özdem for creating the graphics. I also want to thank Ilse Evertse and her associates for the editing of this book

I hope that you enjoy reading about, getting to know the many new aspects of Internet computing, and experiencing its fascination. We are responsible for the Internet's future—not only by providing its technological advances but also by finding innovative ways of generating value for individuals, organizations, and societies alike. Let us take on this responsibility together.

Last but not least, the author is responsible for all the formal and content errors.

Karlsruhe, Germany

Ali Sunyaev

Contents

1	Introduction to Internet Computing	1
1.1	A Brief History of the Internet	2
1.1.1	Phase I: Development of Technological Fundamentals	2
1.1.2	Phase II: Growth and Internationalization	4
1.1.3	Phase III: Commercialization and the World Wide Web	7
1.2	Defining Internet Computing	8
1.2.1	Applications	11
1.2.2	Architectures	12
1.2.3	Technologies	13
1.2.4	Systemic Matters	15
1.3	Distributed Information Systems for Internet Computing	16
1.3.1	Distributed Systems	16
1.3.2	Information Systems	16
1.3.3	Design Challenges of Distributed Information Systems	17
1.4	Application Examples of Internet Computing	19
2	Information Systems Architecture	25
2.1	Defining Information Systems Architecture	26
2.2	The Principles of Information System Architecture	27
2.2.1	Principle 1: Architecture Models Information System Boundaries, Inputs, and Outputs	28
2.2.2	Principle 2: An Information System Can Be Broken down into a Set of Smaller Subsystems	28
2.2.3	Principle 3: An Information System Can Be Considered in Interaction with Other Systems	29
2.2.4	Principle 4: An Information System Can Be Considered Through Its Entire Lifecycle	30

2.2.5	Principle 5: An Information System Can Be Linked to Another Information System via an Interface	30
2.2.6	Principle 6: An Information System Can Be Modeled at Various Abstraction Levels	31
2.2.7	Principle 7: An Information System Can Be Viewed Along Several Layers	32
2.2.8	Principle 8: An Information System Can Be Described Through Interrelated Models with Given Semantics . . .	32
2.2.9	Principle 9: An Information System Can Be Described Through Different Perspectives	33
2.3	Architectural Views	34
2.4	Architectural Patterns	35
2.4.1	Client-Server Architectures	36
2.4.2	Tier Architectures	37
2.4.3	Peer-to-Peer Architectures	43
2.4.4	Model View Controller Architectures	44
2.4.5	Service-Oriented Architecture	45
3	Design of Good Information Systems Architectures	51
3.1	Architecture Design	52
3.2	IS Architectures' Quality	55
3.2.1	Functional and Nonfunctional Requirements	56
3.2.2	Quality Attributes	57
3.3	The Information Systems Architecture Design Process	66
3.3.1	Basic Process Activities	66
3.3.2	Example Method for Designing Architectures: Attribute-Driven Design (ADD) Method	73
3.3.3	Success of Architecture Design Processes: The Iron Triangle	75
4	Internet Architectures	83
4.1	History of the Internet	84
4.2	Today's Internet Network Infrastructure	86
4.3	The Internet Protocol	92
4.3.1	Internet Protocol Suite	92
4.3.2	IP Addresses	97
4.3.3	Domain Name System	98
4.3.4	IP-Routing and Packet Forwarding	101
4.4	Content Delivery Networks	105
4.5	Emerging Internet Network Architecture	109
4.5.1	Software-Defined Networking	109
4.5.2	Network Functions Virtualization	112
4.5.3	Overlay Networks	114
4.5.4	Information-Centric Networking	115

5	Middleware	125
5.1	Introduction to Middleware	126
5.2	Remote Procedure Call	129
5.3	Middleware Categories	133
5.3.1	Message-Oriented Middleware	133
5.3.2	Transaction-Oriented Middleware	139
5.3.3	Object-Oriented Middleware	144
6	Web Services	155
6.1	Introduction to Web Services	156
6.2	Basic Web Technologies	159
6.2.1	Hyper Text Transfer Protocol (HTTP)	159
6.2.2	Extensible Markup Language (XML)	163
6.3	Web Service Architectures	171
6.3.1	Service-Oriented Architecture	171
6.3.2	Internal and External Web Service Architecture Perspectives	174
6.3.3	SOAP Web Services	176
6.3.4	RESTful Web Services	184
6.3.5	Differentiating Between RESTful and SOAP Web Services	189
7	Cloud Computing	195
7.1	An Introduction to Cloud Computing	196
7.1.1	The Emergence of Cloud Computing	196
7.1.2	Definition of Cloud Computing and its Essential Characteristics	198
7.1.3	The Cloud Service Market	200
7.1.4	Cloud Computing Service Models	203
7.1.5	Cloud Computing Deployment Models	205
7.1.6	Differences Between Related Concepts	208
7.2	Essentials to the Provision of Cloud Services	210
7.2.1	Essential Cloud Technologies	210
7.2.2	Cloud Service Stack	212
7.3	Chances and Challenges of Cloud Computing	214
7.3.1	Reasons to Move into the Cloud: Benefits and Opportunities for Organizations	214
7.3.2	Cloud Computing's Transformative Mechanisms	218
7.3.3	The Downside of Cloud Computing: New Risks and Challenges	220
7.4	Security and Data Protection in Cloud Environments	223
7.4.1	Security and Privacy Challenges Due to Essential Cloud Service Characteristics	223
7.4.2	Continuous Service Certification as Innovative Means to Ensure Security and Data Protection	226

8	Fog and Edge Computing	237
8.1	Fog and Edge Computing Fundamentals	238
8.1.1	Definition and Characteristics of Fog Computing	239
8.1.2	Fog Computing Service Models	245
8.1.3	Fog Computing Deployment Models	246
8.1.4	Definition and Characteristics of Edge Computing	247
8.1.5	Mist Computing	249
8.1.6	Differences to Cloud Computing	249
8.2	Challenges and Opportunities of Fog and Edge Computing	252
8.2.1	Challenges of Fog and Edge Computing	252
8.2.2	Opportunities	256
8.3	Fog and Edge Computing in Practice	259
8.3.1	OpenFog Reference Architecture for Fog Computing	259
8.3.2	Video Analytics	259
8.3.3	Augmented Reality Glasses	260
9	Distributed Ledger Technology	265
9.1	Background of Distributed Ledger Technology	266
9.1.1	Distributed Ledger Technology as a Game Changer	266
9.1.2	History of Distributed Ledger Technology	271
9.1.3	Terminology in Distributed Ledger Technology	274
9.2	Technical Foundations	276
9.2.1	Hash Functions	277
9.2.2	Merkle Tree	278
9.2.3	Public Key Infrastructure	279
9.2.4	Consensus Mechanisms in Distributed Ledger Technology	281
9.3	The Bitcoin Blockchain	284
9.4	Smart Contracts	289
9.5	Applications of Distributed Ledger Technology	290
9.5.1	Financial Technology	291
9.5.2	Health Care	292
9.5.3	Supply Chain Management	292
10	The Internet of Things	301
10.1	Introduction of the Internet of Things	302
10.1.1	Definition and Characteristics	302
10.1.2	A Brief History of the Internet of Things	304
10.2	The Internet of Things: Technologies and Architectures	306
10.2.1	Enabling Technologies	306
10.2.2	Core Concepts	310
10.2.3	Architecture Models	313
10.3	Internet of Things Applications	318
10.3.1	Smart Homes	318
10.3.2	Smart Cities	321

10.3.3	The Industrial Internet of Things	324
10.3.4	The Internet of Things in the Energy and Health Care Sectors	327
10.4	Challenges and the Future of the Internet of Things	329
10.4.1	Challenges	329
10.4.2	Outlook: The Future of the Internet of Things	332
11	Critical Information Infrastructures	339
11.1	Foundations of Critical Information Infrastructures	340
11.1.1	The Emergence of Critical Information Infrastructures	340
11.1.2	Sociotechnical Systems	342
11.1.3	Conceptualization of Critical Information Infrastructures	345
11.1.4	Differences Between Critical Infrastructures and Critical Information Infrastructures	346
11.2	Properties of Critical Information Infrastructures	349
11.3	Functions of Critical Information Infrastructures	353
11.3.1	Communication	353
11.3.2	Governance	355
11.3.3	Knowledge Management	357
11.3.4	Information Collection	359
11.4	Operation of Critical Information Infrastructures	360
12	Emerging Technologies	373
12.1	Emergence and Emerging Technology	374
12.2	Immersive Technologies	381
12.2.1	Virtual Reality	383
12.2.2	Augmented Reality	386
12.3	Virtual Assistant	390
12.4	Artificial Intelligence	394
	Glossary	407

Abbreviations

ADD	Attribute-driven design
AED	Automated external defibrillator
AFRINIC	African Network Information Center
AGI	Artificial general intelligence
AI	Artificial intelligence
Amazon S3	Amazon Simple Storage Service
AMQP	Advanced Message Queuing Protocol
ANI	Artificial narrow intelligence
API	Application programming interface
APNIC	Asia-Pacific Network Information Centre
AR	Augmented reality
ARIN	American Registry for Internet Numbers
ARP	Address Resolution Protocol
ARPA	Advanced Research Project Agency
ASI	Artificial super intelligence
ASR	Architecturally significant requirements
AV	Augmented virtuality
AWS	Amazon Web Services
B2B	Business-to-business
B2C	Business-to-consumer
BBN	Bolt, Beranek, and Newman
BOT	Beginning of transaction
BTC	Bitcoin
CA	Certification authority
CAVE	Cave automatic virtual environment
CCN	Content-centric networking
CDC	Centers for Disease Control and Prevention
CDN	Content delivery network
CGM	Continuous glucose monitoring
CICS	Customer Information and Control Systems
CII	Critical information infrastructure

CIX	Commercial Internet eXchange
CLR	Common Language Runtime
COM	Component Object Model
COMPAS	Correctional Offender Management Profiling for Alternative Sanctions
CORBA	Common Object Request Broker Architecture
CRL	Certification revocation list
CRM	Customer relationship management
CSC	Cloud service certification
CSNET	Computer Science Network
CSS	Cascading Style Sheets
DARPA	Defense Advanced Research Projects Agency
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DCPS	Data-Centric Publish/Subscribe
DDoS	Distributed denial of service
DII	Dynamic Invocation Interface
DLRL	Data Local Reconstruction Layer
DLT	Distributed Ledger Technology
DNS	Domain name system
DOE	Department of Energy
DoS	Denial-of-Service
DPoS	<i>Delegated Proof-of-Stake</i>
DRE	Direct-recording electronic
DSI	Dynamic Skeleton Interface
EAI	Enterprise Application Integration
EAS	Emergency Alert System
EHR	Electronic health record
EOT	End of transaction
EPA	Environmental Protection Agency
ESIOP	Environment-Specific Inter-ORB Protocol
ESNet	Energy Sciences Network
ETSI	European Telecommunications Standards Institute
EU	European Union
EVM	Ethereum Virtual Machine
FCC	Federal Communications Commission
FDDI	Fiber Distributed Data Interface
FEMA	Federal Emergency Agency
Fid	Forwarding identifier
FTP	File Transfer Protocol
GIOP	General Inter-ORB Protocol
GP	General practitioner
GPS	Global Positioning System
GTP	Game transfer phenomena

HLL	High-level programming language
HMD	Head-mounted display
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
IAB	Internet Architecture Board
ICANN	Internet Corporation for Assigned Names and Numbers
ICCC	International Computer Communications Conference
ICMP	Internet Control Message Protocol
ICN	Information-centric networking
ICO	Initial coin offering
ICT	Information and communication technology
IDE	Integrated development environment
IDL	Interface definition language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IGRP	Interior Gateway Routing Protocol
IIN	Interbank Information Network
IIOF	Internet Inter-ORB Protocol
IMP	Interface Message Processor
IoT	Internet of Things
IP	Internet Protocol
IPTO	Information Processing Technique Office
IRTF	Internet Research Task Force
IS	Information systems
ISO	International Organization for Standardization
ISOC	Internet Society
ISP	Internet Service Provider
IT	Information technology
ITU	International Telecommunication Union
IXP	Internet Exchange Point
J2SE	Java 2 Standard Edition
Java EE	Java Platform Enterprise Edition
JMS	Java Messaging Service
JSON	<i>JavaScript Object Notation</i>
LACNIC	Latin America and Caribbean Network Information Centre
LAN	Local Area Network
LBS	location-based service
LCD	Liquid crystal display
LPC	Local procedure call
MAE	Metropolitan Area Exchange
MOM	Message-oriented middleware
MQTT	Message Queuing Telemetry Transport
MR	Mixed reality

MTBF	Mean time between failure
MTS	Microsoft Transaction Server
MTTR	Mean time to repair
MTU	Maximum transmission unit
MVC	Model View Controller
NAP	Network Access Point
NASA	National Aeronautics and Space Administration
NCP	Network Control Program
NFS	Network File System
NFV	Network function virtualization
NIST	National Institute of Standards and Technology
NOAA	National Oceanic and Atmospheric Administration
NPL	National Physical Laboratory
NSF	National Science Foundation
OMG DDS	Data Distribution Service by the Object Management Group
OOM	Object-oriented middleware
ORB	Object Request Broker
OSI	Open Systems Interconnection
P2P	Peer-to-peer
PaaS	Platform as a Service
PBFT	Practical Byzantine fault tolerance
PC	Personal Computer
PDA	Personal digital assistant
PIT	Pending interest table
POA	Portable Object Adapter
POP	Point of presence
POP3	Post Office Protocol 3
PoS	<i>Proof-of-Stake</i>
PoW	Proof of Work
PSIRP	Publish-Subscribe Internet Routing Paradigm
PSMOM	Publish/subscribe message-oriented middleware
QoS	Quality of service
RA	Registration authority
RARP	Reverse Address Resolution Protocol
REST	Representational State Transfer
RFC	Request for Comments
RFID	Radio frequency identification
RId	Rendezvous identifier
RIP	Routing Information Protocol
RIPE NCC	Réseaux IP Européens Network Coordination Centre
RIR	Regional Internet Registries
RMI	Remote Method Invocation
RSM	<i>Replicated state machine</i>
SaaS	Software as a Service

SAML	Security Assertion Markup Language
SCADA	Supervisory control and data acquisition
SCTP	Stream Control Transmission Protocol
SDN	Software-defined networking
SGML	Standard Generalized Markup Language
SHA	Secure hash algorithm
SIId	Scope identifier
SLA	Service-level agreements
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
SQS	Amazon Simple Queue Service
SRI	Stanford Research Institute
SSL	Secure Sockets Layer
STOMP	Streaming Text-Oriented Messaging Protocol
SUS	System Usability Scale
TCP	Transmission Control Protocol
TDAG	Transaction-based directed acyclic graphs
TIP	Terminal Interface Processor
TLS	Transport Layer Security
TOM	Transaction-oriented middleware
TP	Transaction processing
TRPC	Transactional remote procedure call
TTL	Time to live
UCLA	University of California, Los Angeles
UCSB	University of California, Santa Barbara
UDP	User Datagram Protocol
UID	Unique identifier
UML	Unified Model Language
URI	Uniform Resource Identifiers
UTXO	Unspent transaction output
VC	Venture capital
VNF	Virtualized network function
VoIP	Voice over IP
VPN	Virtual private network
VR	Virtual reality
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WAL	Write-ahead log
WAN	Wide Area Network
WCF	Windows Communication Foundation
WHO	World Health Organization
WSAN	Wireless sensor and actuator network

WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
WSN	Wireless sensor network
WWW	World Wide Web
XaaS	Everything as a Service
XML	Extensible Markup Language
XSD	XML Schema Definition

Chapter 1

Introduction to Internet Computing



Abstract

Over the past decades, the Internet has fundamentally influenced almost all areas of our everyday lives. It has profoundly changed the ways in which we communicate, gather information, and consume media, and has led to the emergence of Internet companies that are based on fundamentally new business models. This chapter introduces Internet computing as a scientific field that is concerned with applications provided via the Internet, the underlying architectures and technologies necessary to build such applications, and systemic matters that inform the design of such applications. Based on these foundations, this chapter outlines this book's structure. In addition to defining Internet computing and briefly presenting the chapters, an overview of the historical background and development of the Internet is provided. This chapter also introduces the concepts of information systems (IS) and distributed systems as important related scientific fields that shaped the ways Internet-based applications have been designed. To round off this introduction, several common Internet-based applications are presented.

The Learning Objectives of this Chapter

This chapter's main learning objective is to provide a basic understanding of the concept of Internet computing. This chapter also gives readers a brief impression of the major contents of this book and how it is structured. After having read this chapter, readers will understand the history and current developments of the Internet. Readers will also learn about the scientific fields of information systems and distributed systems and how they shaped the ways we think about the design of Internet-based applications. Finally, readers are presented with examples of Internet-based applications and learn more about their relevance in our daily private and business lives.

The Structure of this Chapter

This chapter is structured as follows: Section 1 briefly discusses the relevance of the Internet for our private, work, and societal lives and outlines the history of the Internet. Section 2 presents a definition of the scientific field of Internet computing, followed by an overview of the book's structure and a brief outline of each chapter. Section 3 deals with the related scientific fields of IS and distributed systems and

briefly explains why they are important for Internet computing. The chapter concludes with a description of typical examples of Internet-based applications.

1.1 A Brief History of the Internet

Today, we can hardly imagine life without the Internet. A recent study reports that Americans spend 24 hours per week online (Cole et al. 2018). The most important activities people do online include direct communication (e.g., e-mails, instant messaging, or phone calls), social networking, information gathering, media streaming, and gaming. But the Internet has become indispensable, and not only for private use. It has also become indispensable to companies and organizations of all kinds, and substantially influences societies as a whole. In sum, the Internet has brought significant changes to all areas of our daily lives. In an organizational context, the Internet first became relevant for traditional companies, which used the Internet primarily as a platform for communication and marketing. However, the Internet has also fueled the development of a variety of new business models that would have been impossible without it. Famous companies that have grown on the basis of such business models include Amazon, Google, and Facebook. Concerning the Internet's impacts on our private and business realities as well as its omnipresence, it is astonishing to consider that the Internet as we know it has not existed for very long.

In this chapter, the Internet's history is briefly traced along the most important developments and technological achievements. Its history can be divided into three main phases: (1) the development of the technological fundamentals of the Internet, from the mid-1960s, (2) the growth and internationalization of the Internet, from the mid-1970s, and (3) the commercialization of the Internet, from the early 1990s. Each of these phases will be elaborated on in the following sections.

1.1.1 Phase I: Development of Technological Fundamentals

While it is hard to determine exactly when the history of the Internet started, October 4, 1957 is a suitable candidate. During this time, the Cold War was in full swing and a race in (nuclear) armament and technological development between two competing blocs, East and West, had begun. Each technological advantage of the opposing side was perceived as a potential threat to one's own security. For this reason, the U.S. was shocked when the USSR, on October 4, 1957, was the first country to successfully launch an artificial earth satellite, Sputnik 1, into orbit. In direct reaction, on February 7, 1958 the U.S. founded the Advanced Research Project Agency (ARPA). ARPA's purpose was to form and execute research projects to expand the frontiers of technology and science. In October 1962, J.C.R. Licklider

became the first head of the Information Processing Techniques Office (IPTO) at ARPA. Licklider, a former MIT professor, had just published a paper discussing a *Galactic Network* concept through which everyone could quickly access data and programs from anywhere (Licklider and Clark 1962). In spirit, the concept had strong similarities to the Internet as we know it (Leiner et al. 2009). At about the same time, a paradigm shift from circuit-oriented to packet switching concepts took place in telecommunications (the different switching concepts are explained in detail in Chapter 4). This shift was mainly driven by Leonard Kleinrock at MIT, Paul Baran at RAND Corporation, and Donald Davies at the National Physical Laboratory (NPL) in England (Kleinrock 1961, 2007; Baran 1964). Fundamentally, it involved the separation of communications into small data packets that, provided with destination and sender addresses, autonomously find their way through a network – a prerequisite for the distributed, decentralized architecture of the Internet. Interestingly, the work at MIT, RAND, and NPL had initially all proceeded in parallel without any of the researchers knowing about one another’s work (Leiner et al. 2009).

In 1966 Robert Taylor, a former NASA employee, was promoted to the head of IPTO at ARPA. As Licklider had convinced him of the importance of his vision, Taylor sought to realize Licklider’s ideas of an interconnected networking system. To do so, he initiated the ARPANET project and recruited Lawrence G. Roberts from MIT as the program manager. At the 1967 ACM Symposium on Operating System Principles, Roberts presented a first concept of ARPANET (Roberts 1967). There, he met Roger Scantlebury, a member of Davies’s team at NPL, who presented their research on packet switching and suggested it for use in ARPANET. Roberts decided to apply packet switching for ARPANET and subsequently further refined the overall structure.

ARPA invited external tenders for various components of the new network. The Stanford Research Institute (SRI) was commissioned to write the specifications of the network, the network measurement system was prepared by Leonard Kleinrock’s team at University of California, Los Angeles (UCLA), and the company Bolt, Beranek and Newman (BBN) developed packet switching techniques, specifically an Interface Message Processor (IMP). Later, Robert E. Kahn, a BBN employee, would play a major role in the overall ARPANET architectural design. In the fall of 1969, the first four computers were connected at UCLA, SRI, the University of California at Santa Barbara (UCSB), and the University of Utah (see Fig. 1.1). On October 29, 1969, “I” and “O” were the first successful messages that UCLA sent to SRI via ARPANET. When Kleinrock’s team at UCLA sought to send a third message, the system crashed.

ARPANET’s development centered around the Request for Comments (RFC) process, which is still used today for proposing and distributing Internet protocols and systems. In RFCs, the technical standards of ARPANET were not enacted in the style of a law, but as a friendly request for comments. Steve Crocker, the author of the first RFC (Crocker 1969), explained this approach via the fact that many

participants were doctoral students with no authority. Thus, they had to find a way to document their work and invite others for discussion without seeming to impose anything on anyone. RFCs can generally be created by anyone. They are intended as discussion papers, with the stated goal of breaking the authority of what is written.

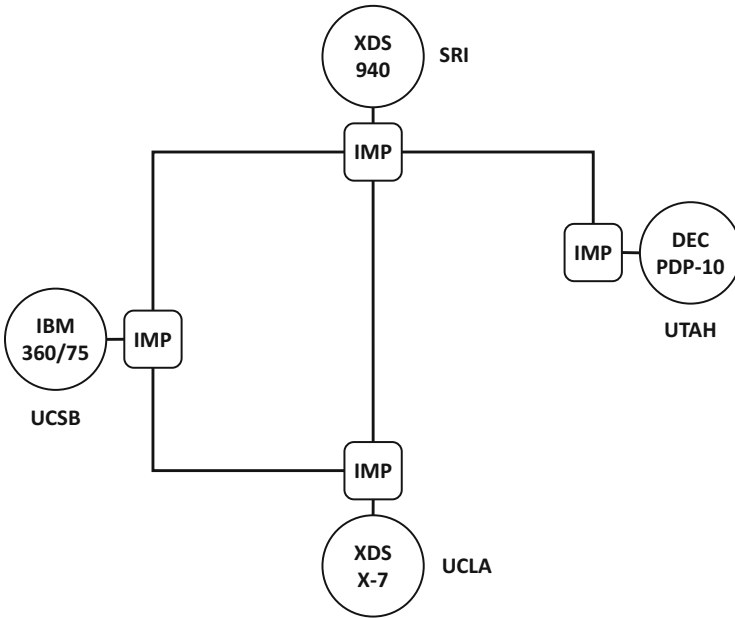


Fig. 1.1 The initial four-node ARPANET in 1969 (adapted from Kleinrock (2010))

1.1.2 Phase II: Growth and Internationalization

In the following years, ARPANET grew rapidly. By December 1970, it contained 13 nodes, with one additional node added approximately every month (see Fig. 1.2). In 1972, ARPA changed its name to Defense Advanced Research Projects Agency (DARPA). In the same year, Lawrence G. Roberts (MIT) decided to ask Robert E. Kahn at BBN to organize a first public demonstration of ARPANET. In October 1972, this took place at the International Computer Communications Conference (ICCC). In the basement of the Washington Hilton Hotel, a packet switching computer and a Terminal Interface Processor (TIP) were installed. It was connected to 40 machines across the U.S. Demonstrations included interactive chess games and the simulation of an air traffic control system. Overall, the demonstration was a huge success.

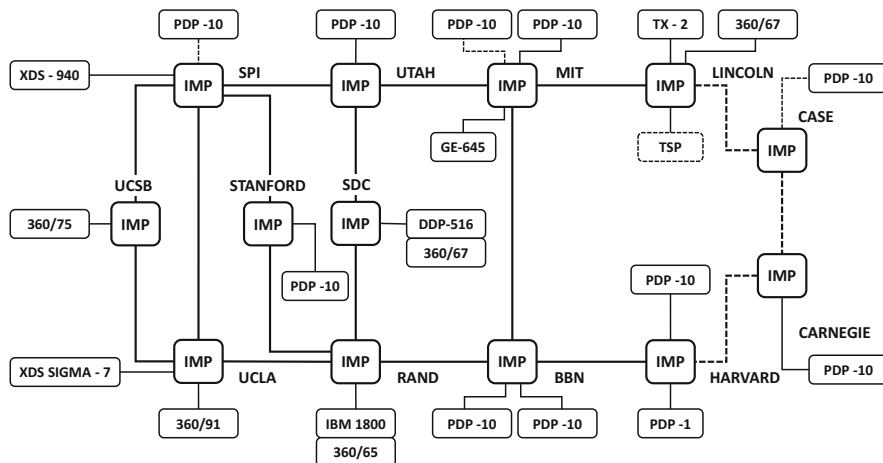


Fig. 1.2 ARPANET in December 1970 (adapted from Bolt Beranek and Newman (1981))

In the 1970s, computer networks similar to ARPANET (e.g., the NPL network in the UK and the CYCLADES network in France) were installed across the world. Every network had its own specification and, to connect them, something was needed to unify them. Robert E. Kahn, who had switched from BBN to DARPA asked Vinton Cerf of Stanford University to work with him on this problem. They defined a fundamental reformulation, in which the differences between network protocols were hidden by using a shared internetwork protocol. Their first version of the internetwork Transmission Control Protocol (TCP/IP) was presented in RFC 675 in December 1974. It was also in RFC 675 that the word *Internet* was first used as a shorthand for *internetworking*. The development of TCP/IP enabled one to connect almost any networks, no matter what their characteristics were. However, implementing TCP/IP in ARPANET was a tough task, and ARPANET did not change its internal core network protocols from Network Control Program (NCP) to TCP/IP until 1983, an event that was later referred to as Flag Day. After several years of work, on November 22, 1977 a three-network demonstration was conducted that included ARPANET, the SRI's Packet Radio Van on the Packet Radio Network, and the Atlantic Paket Satellite network. This marked the start of long-term experimentation and development to evolve and mature Internet concepts and technologies. Starting with these three networks, the experimental environment grew and incorporated essentially every form of network and a very broad-based research and development community – the Internet was born!

After ARPANET had been running for several years, DARPA looked for another agency to hand over the network to. DARPA's primary purpose was funding and research, not the running of a communication network. In July 1975, ARPANET was turned over to the Defense Communications Agency, which was also part of the Department of Defense. However, networks based on ARPANET were still government-funded and were therefore restricted to noncommercial uses, such as

research. Unrelated commercial use was strictly forbidden. This initially restricted connections to military sites and universities. During the 1980s, the connections expanded to more educational institutions, and even to a growing number of companies such as Digital Equipment Corporation and Hewlett-Packard, which were participating in research projects or were providing services to these research institutions. Several other branches of the U.S. government – such as the National Aeronautics and Space Administration (NASA), the National Science Foundation (NSF), and the Department of Energy (DOE) – began to develop a successor to ARPANET. In the mid-1980s, all three of these branches developed Wide Area Networks (WANs) based on TCP/IP. NASA developed the NASA Science Network, NSF developed CSNET, and the DOE evolved the Energy Sciences Network (ESNet). Similarly, outside the U.S., various organizations began to develop their own networks. For instance, the European Organization for Nuclear Research CERN began installing and operating TCP/IP to interconnect its major internal computer systems, workstations, PCs, and an accelerator control system. CERN continued to operate a limited self-developed system (CERNET) internally and several incompatible network protocols externally. However, there was considerable resistance in Europe to a more widespread use of TCP/IP. Instead, European institutions were more convinced of X.25 networks and the Open Systems Interconnection (OSI) standard, which was developed by the International Organization for Standardization (ISO) since 1982, a development that would later not prevail.

When desktop computers first appeared, some thought that TCP/IP was too big and too complex to run on personal computers. David Clark and his research group at MIT set out to show that a compact and simple implementation of TCP/IP was possible (Speed et al. 2001). They developed implementations for the Xerox Alto and the IBM PC. Their implementations were fully interoperable with other TCPs and showed that workstations as well as large time-sharing systems could form part of the Internet. Widespread development of Local Area Networks (LANs), PCs, and workstations in the 1980s allowed the Internet to grow further. Thus, by the mid-1980s, the Internet had already become a well-established technology to support a broad community of researchers and developers. Also, it was beginning to be used by other communities for daily computer communications. Electronic mail was being used broadly across several communities, often with different systems. The interconnection of different mail systems was showing the utility of inter-personal electronic communication (Leiner et al. 2009).

In 1986, the NSF created NSFNET, a 56 kbit/s backbone to support the NSF-sponsored supercomputing centers. Because the 56 kbit/s network quickly became overloaded, NSFNET was upgraded to 1.5 Mbit/s in 1988 and to 45 Mbit/s in 1991. The NSF decided to make TCP/IP mandatory for the program. NSFNET was interlinked with ARPANET and allowed ARPANET to be decommissioned in 1990. The network continued to spread among research and academic institutions throughout the U.S., including connections to research networks in Canada and Europe, greatly extending the Internet's size and reach. At this time, TCP/IP had supplanted or marginalized most other wide-area computer network protocols worldwide, and TCP/IP was well on its way to becoming the bearer service for the Global Information Infrastructure (Bauer and Latzer 2016).

1.1.3 Phase III: Commercialization and the World Wide Web

A turning point toward the commercialization of the Internet was the shutdown of ARPANET in 1990. While originally any commercial use of the network was prohibited, the exact definition of commercial use of ARPANET was unclear and subjective. With advancing technological developments, more and more stakeholders became interested in exploiting the potentials of such commercial use. In contrast to ARPANET, other related networks had no such restrictions. Thus, connections of these networks to NSFNET were officially blocked. In 1992, a first step toward opening the NSFNET for commercial use was taken when the U.S. Congress passed the Scientific and Advanced-Technology Act, 42 U.S.C. §1862(g), which allowed the NSF to support access by the research and education communities to computer networks that were not used exclusively for research and education purposes. Internet Service Providers (ISPs) – including PSINet, Altnet, CERFNet, and UUNET – were formed to provide network access to commercial customers. Thus, NSFNET was no longer the de facto backbone and exchange point of the Internet. The Commercial Internet eXchange (CIX), Metropolitan Area Exchanges (MAEs), and later Network Access Points (NAPs) were becoming the primary interconnections between many networks. The final restrictions on carrying commercial traffic ended on April 30, 1995, when the NSF ended its sponsorship of the NSFNET Backbone Service and the service ended.

The growing mass of information available on the Internet became increasingly hard to handle, and many people realized the need to be able to organize and find relevant information. Thus, a key driver in the commercialization of the Internet was the invention of the World Wide Web (WWW), which – as is explained later – helped to overcome these challenges. The development of the WWW goes back to Tim Berners-Lee, who worked at CERN. By combining existing technologies such as hypertext, the Internet, and multifont text objects, he was able to build the necessary tools by the end of 1990: the HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), the first Web browser (called WorldWideWeb, later renamed Nexus), the first HTTP server software, the first Web server, and the first Web pages that described the project itself. However, early adopters of the Web were primarily university-based scientific departments or physics laboratories. One problem was that a Web browser was initially only available for the NeXT operating system. A boost in the spread of the Web was triggered by the development of the Web browser Mosaic, a graphical browser that eventually ran on several popular office and home computers. It was the first web browser with the aim to bring multimedia content to nontechnical users, and therefore included images and text on the same page, unlike previous browser designs. Its development goes back to Marc Andreessen and his team at the University of Illinois. Mosaic was later further developed and renamed Netscape.

In the following years, the Internet increased in popularity owing to increasing connection speed and lower fees. As a result, the Internet became more commercially attractive for companies, because millions of potential customers were now

much easier to reach. Companies of all kinds hurried to set up websites. Some entrepreneurs went further and set up companies that only operated on the Internet and offered goods and services solely on it. With little seed capital, they were able to implement ideas that were well received by customers. To further expand their businesses, they raised additional capital by going public – a development that in retrospect is referred to as the dot-com bubble. The dot-com bubble ended in 2000 with a stock market crash, which heralded the start of a general downward trend in the world's stock markets. Many newly created Internet companies had to close. However, established Internet companies such as Amazon and Google were not endangered, and the Internet boom continued, although at a slower pace.

Early in its history, the Web was primarily used to present digitized information that previously had been presented in an analog way. Thus, the first phase of the Web was characterized by static low-interaction content such as personal websites. What followed was the era of Web 2.0. The term was originally coined by Darcy DiNucci in 1999 (DiNucci 1999). Web 2.0 refers not to an update to any technical specification, but to changes in the ways Web pages were designed and used. It is characterized especially by the change from static web pages to dynamic and/or user-generated content and the growth of social media. Web 2.0 describes a change in which the World Wide Web became an interactive experience between users and Web publishers, rather than the one-way conversation that had previously existed (Laudon and Laudon 1999). It also represents version of the Web, where nearly everyone is able to contribute and participate, despite their limited technical knowledge. Today, the Web has evolved beyond Web 2.0, and new concepts such as Web 3.0 and Web 4.0 are being discussed. The term Web 3.0 (also called Semantic Web) was coined by the inventor of the original Web technologies, Tim Berners-Lee. In particular, it refers to a Web of data in which the meaning of such data is machine-readable across all Web participants, for instance, by providing metadata. Various different well-established standards such as Extensible Markup Language (XML) exist that support Web 3.0 features. Unlike its predecessors, Web 4.0 has no established uniform meaning. The term primarily refers to the progressive fusion of Web technologies with other current computing trends such as augmented reality (AR) and artificial intelligence (AI). It is assumed that by doing so, the borders between the real world and the virtual world are blurring, leading to an omnipresence of the Web in our everyday lives and the emergence of new concepts such as the Internet of Things (IoT). The ever-continuing discussion of new Web trends shows that the evolution of the Internet and the Web have led to profound changes in our daily private and business lives.

1.2 Defining Internet Computing

The Internet has become a major part of our economic, scientific, and social lives. As its history shows, the Internet landscape is evolving and is becoming increasingly complex, in terms of its underlying infrastructure as well as in terms of its more

widespread use by organizations and individuals owing to the development of new and innovative tools. This book provides insights into the most important technologies and concepts relating to the Internet. The scientific domain concerned with these topics is Internet computing. To establish a shared terminology for the subsequent chapters, this term will now be defined.

It is hard to derive a suitable definition of *Internet computing*, because its boundaries are vague and constantly evolving, as depicted by the recent – rapid – development of the Internet. In 2005, Robert E. Filman, a former Editor In Chief of the scientific journal *IEEE Internet Computing*, wrote in an editorial:

“Biology is the study of living organisms’ is a fine definition because the domain is both fairly well delimited and has a certain uniformity. All living things reproduce and decrease entropy, and mechanisms such as DNA, metabolism, and evolution find themselves applied consistently to the domain of living things. Saying that Internet computing is ‘the study of anything having to do with the Internet’ is clearly less satisfactory. It is a definition tied too closely to a particular moment in technology. When computing mechanisms have pervasively spread to every doorknob and plumbing fixture, computers have become as embedded in the natural fabric of economic existence as printed language, and interconnectivity is as omnipresent as radio reception, will it still be Internet computing?” (Filman 2005)

This emphasizes two things: (1) Filman is an expert in the field, who could anticipate the Internet’s ubiquitousness even before smartphones became mainstream with the first iPhone in 2007, and (2) Internet computing is a very broad and complex phenomenon. Particularly owing to these two characteristics, the Internet’s ever-evolving nature as well as its complexity and diversity, it is unfeasible to capture all aspects of Internet computing in this book. Instead, this book seeks to give basic insights into the fundamental concepts of Internet computing and to provide a selection of current and emerging Internet computing trends. This is particularly important because, as Munindar P. Singh – a predecessor of Filman at *IEEE Internet Computing* – stated, in his *Practical Handbook on Internet Computing*:

“[T]he sheer diversity of the topics that Internet computing integrates has also led to a situation where most of us have only a narrow understanding of the subject matter as a whole. Clearly, we cannot be specialists in everything, but the Internet calls for a breed of super-generalists who not only specialize in their own area, but also have a strong understanding of the rest of the picture.” (Singh 2004)

In accordance with the literature, Internet computing is not limited to the public Internet as most of us experience it daily (Singh 2004). In contrast, it also applies to Internet-based applications that are used in a single organization or in a closed group of organizations. In our globalized (business) world, this is particularly important, since the Internet’s infrastructure now allows users to easily bridge large geographical distances, for instance, between employees of a globally distributed firm or a firm and its suppliers. Instead of building expensive proprietary solutions that could similarly allow this type of communication, organizations can build up on the Internet infrastructure and on the various different best practices that have been built around or within it. Thus, the benefits of a global high-speed communication

utility become accessible to everyone – from small startups to large global companies.

In this book, Internet computing is conceptualized along four major aspects: (1) applications, (2) architectures, (3) technologies, and (4) systemic matters. First, *applications* refer to the variety of Internet-based applications that are provided via the Internet. These can target a wide range of different user groups (e.g., services targeted at a broad mass of consumers or tailored solutions for a specific customer). Second, *architectures* lay the foundation for such applications and describe the basic components and structures that allow Internet-based applications to work. They include both considerations about the technical infrastructure of the Internet and structural considerations about the design of software and systems. Third, *technologies* denote the underlying technical capabilities of the Internet that enable the development of such applications. They describe the entities (both material and immaterial) to solve specific real-world problems concerned with Internet-based applications. Finally, *systematic matters* include important trends and paradigms that inform the design of Internet-based applications. These include but are not limited to the consideration of ethical aspects (e.g., trust, privacy), or the Internet landscape as a holistic system.

In sum, this book defines Internet computing as concerned with the applications provided on the Internet, the architectures and technologies used in applications on the Internet, and the systemic matters that shape the design of applications on the Internet. Internet computing encompasses all applications irrespective of whether they are built for the general public (e.g., social network services) or are solely used within a single organization (e.g., enterprise resource planning systems) or a closed group of organizations (e.g., supply chain management systems).

Internet Computing

Internet computing is concerned with the *applications* provided on the Internet, the *architectures* and *technologies* used in applications on the Internet, and the *systemic matters* that shape the design of applications on the Internet. Internet computing encompasses all applications irrespective of whether they are built for the general public (e.g., social network services), or solely used in a single organization (e.g., enterprise resource planning systems) or in a closed group of organizations (e.g., supply chain management systems).

To illustrate this definition, Fig. 1.3 depicts the house of Internet Computing. It consists of the three pillars *architectures*, *technologies*, and *systemic matters* that are needed to design and develop good Internet-based applications, and the applications themselves, represented as the roof. The purpose of the house is to guide readers through the book, while some of the most important principles of Internet computing are introduced along the three pillars. The setup of the house in this book will now be explained in some detail.

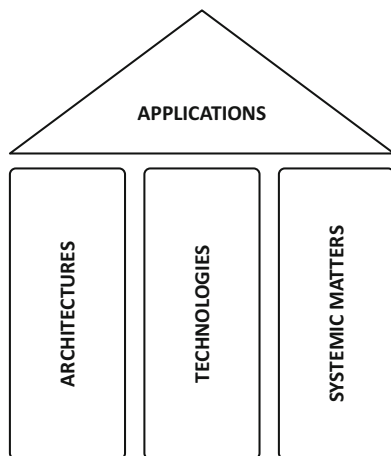


Fig. 1.3 Internet Computing as a House

1.2.1 Applications

As a starting point, Internet-based applications are referred to as applications provided via the Internet. Arguably, such applications can take many different forms and manifestations. To better understand different Internet-based application types, they can be differentiated along various dimensions. They can for instance be distinguished concerning the target group of users they were designed and developed for (e.g., applications designed for the general public vs. applications designed for specific organizations), the intended usage purpose (e.g., private vs. commercial use), or their technological openness (e.g., open-source vs. proprietary applications). This book focuses on Internet-based applications in the form of distributed IS; thus, it examines Internet-based applications mainly from an organizational perspective. Chapter 1.3 explains how distributed IS are related to Internet computing in some detail.

As noted, Internet computing comprises a wide range of possible applications that span different organizations on a global scale. In this book, the goal is not to illustrate this diversity in depth; instead, the focus is on providing a solid foundation for understanding the underlying concepts that allow for this diversity to emerge. Thus, these chapters will provide examples of specific Internet-based applications in order to explain how architectures, technologies, and systemic matters manifest in real-world applications. Some examples of prominent Internet-based applications are introduced in Chapter 1.4.

1.2.2 Architectures

Various definitions of *architecture* exist that emphasize different aspects of architectures across several (scientific) disciplines. Traditionally, the term primarily refers to the process and the product of planning and designing physical buildings. Thus, the *architecture* differs from the *construction* (i.e. the skills and activities of converting architectural considerations into real-world buildings).

Nowadays, architecture not only refers to structural considerations of physical buildings, but is also used in various different contexts, such as enterprise architecture, hardware architecture, or software architecture. In Internet computing, different notions of architectural considerations seem relevant. The chapters in this book that are concerned with architectural considerations of Internet-based applications provide the reader with the tools necessary to identify the key components of Internet-based applications and to design and describe their architectural composition. These four chapters are now briefly introduced.

Information Systems Architectures

Internet-based applications are complex systems that can contain millions of lines of code and various hardware components, and that involve many different types of users and environments. Thus, to enable the design, implementation, and maintenance of such Internet-based applications, we need tools that allow us to capture, describe, and divide them into smaller subsets. For this purpose, we can draw on prior knowledge about the architecture of distributed IS. The chapter *Information Systems Architectures* deals with the principal concepts of IS architecture. It introduces the most common architectural patterns for IS that have proven to be best practices for certain application purposes.

Designing Good Information System Architectures

Owing to Internet-based applications' complexity, the large number of associated design decisions, and the various different requirements often placed on such systems, the design of a good IS is a non-trivial process, which requires much expertise, time, and effort. Thus, the chapter *Designing Good Information System Architectures* takes a process view of IS architecture and sheds light on the process of designing good IS architectures. It outlines which requirements and quality criteria IS architectures should meet, and introduces the major steps in coming up with a suitable architectural design for an IS.

Internet Architectures

The Internet is a massive network of computer networks that connects billions of computers around the globe. Thus, to design and build good Internet-based applications, it is crucial to get a good understanding of what the main components of the Internet are and how they work together in order to make this tremendous network of networks work. The chapter *Internet Architectures* deals with the key components and mechanisms that are necessary to make the Internet possible in its current form. It introduces the Internet's backbone structure, explains the key role of ISPs, and

outlines the essential mechanisms that enable computers to communicate over the Internet.

Middleware

Middleware refers to architectural considerations that seek to utilize application-neutral programs that mediate communication between (Internet) applications in such a way that the complexity of these applications and their underlying operating systems, infrastructures, etc. are hidden. Given Internet-based applications' high complexity and often distributed nature, it is important to understanding how middleware helps to overcome communication barriers between different (organizational) applications (e.g., owing to different operating systems). Thus, the chapter *Middleware* deals with the key aspects of middleware. Three main middleware types are distinguished: (1) message-oriented middleware, (2) transaction-oriented middleware, and (3) object-oriented middleware.

1.2.3 Technologies

There are many definitions and understandings of *technology*. As a starting point, one may refer to the definition by Emmanuel Mesthene from his 1970 book *Technological Change: Its Impact on Man and Society*. He defines technology as “the organization of knowledge for the achievement of practical purposes.” (Mesthene 1970). While this definition remains fairly broad, it has a strong focus on the technology's purpose, which is to solve practical problems. However, when one strictly follows this definition, any knowledge that may in some way contribute to the solution of a practical problem may be classified as technology. Thus, in this book, this definition does not help to delineate the technological aspects and its differences from architectural considerations or systemic matters. A more useful definition and conceptualization is provided by Frederick Ferré in his 1988 book *Philosophy of Technology*. Ferré derives a remarkably precise definition of technology by proposing four major principles: (1) technology is implemented, not empty-handed, (2) technology is practical, not for its own sake, (3) technology is embodied, not in the head alone, and (4) technology is intelligent, not blind. He defines technologies as follows:

“Practical implementations of intelligence (with the caveat that ‘Practical’ requires that they not be wholly ends in themselves; ‘implementations’ entails that a technology be somehow concretely embodied, normally in implements or artifacts, sometimes simply in social organization.)” (Ferré 1988)

In other words, a technology may refer to an (1) implementation used as a means to (2) solve a practical problem that is (3) manifested in the material world as (4) an expression of intelligence (Thierer 2014). In this book, I draw on this conceptualization and refer to technologies as implemented solution approaches that target specific practical problems related to the collection, storage, processing, or

transmission of information. Four chapters deal with specific technologies in the context of Internet computing. They are now briefly introduced.

Web Services

Web services refer to the utilization of open Web standards in order to enable heterogeneous applications from across the world to exchange information, regardless of their hardware configurations, operating systems, or software applications. Thus, Web services are a very popular approach to facilitating automated intra-organizational and inter-organizational communication. The chapter *Web Services* thoroughly introduces the concept of Web services and the different standards and technologies associated with them. It comprehensively introduces the two most important Web technologies that most Web services are based on: HTTP and XML. The two most common Web service variants are then explored in some detail, namely RESTful and SOAP-based Web services.

Cloud Computing

In times of widespread use of the Internet and its services, organizations often need access to large amounts of computing resources while also needing to remain flexible in order to swiftly react to decreasing or increasing demands. As a way to address this issue, cloud computing refers to a technology that enables ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources that can be rapidly provisioned at any time and from any location via the Internet or a similar network (Lins et al. 2015). The chapter *Cloud Computing* introduces the main concepts, explains common service models and deployment models, and outlines the importance of continuous cloud service certification (Lins et al. 2016b; Lins et al. 2016a).

Fog Computing

Owing to upcoming digital innovations such as IoT, embedded artificial intelligent, ubiquitous computing, or 5G wireless networks, millions of new devices and sensors will go online in the next decade. Further, emerging innovations, such as connected vehicles, require real-time processing of data. Thus, the biggest challenges for providers of future Internet-based applications include the management of large amounts of data, sensitive data concerns, and the need for reduced data transfer latencies. To address these challenges and aiming to overcome the shortcomings of pure cloud computing approaches in specific application scenarios (e.g., IoT), fog computing describes a technological paradigm that seeks to move the storage and processing of data to the edge of the network and thus closer to the point where it is needed. The chapter *Fog Computing* thoroughly defines the term and distinguishes it from other related concepts, briefly explains its main challenges and opportunities, and outlines its major contexts of application.

Distributed Ledger Technology

The rise of the cryptocurrency Bitcoin laid a foundation for the emergence of Distributed Ledger Technology (DLT), which has become one of the most hyped information technologies of the past decade. Basically, DLT enables the operation and synchronization of an immutable, append-only distributed database under

consideration of network failures and malicious intents of users of the distributed database. Through the application of cryptographic techniques, DLT seeks to do away with the need for trusted third parties (intermediaries), who are no longer required to prove the authenticity of stored data. The chapter *Distributed Ledger Technologies* briefly outlines the historical development of DLT, introduces its technical foundations, illustrates its functioning with the example of the Bitcoin blockchain, and discusses selected usage cases for the application of DLT.

1.2.4 Systemic Matters

As noted, systemic matters of Internet computing comprise important trends and paradigms that inform the design of Internet-based applications. These include the consideration of ethical aspects (e.g., trust, privacy), or the Internet landscape as a holistic system. While the former is primarily covered implicitly in this book (e.g., when I discuss quality criteria of IS architecture), the latter is explicitly covered by two chapters on Internet computing at a larger scale and the environmental aspects in such contexts that shape the design of Internet-based applications. The two chapters *Internet of Things* and *Critical Information Infrastructures* are now briefly described.

Internet of Things

The Internet has fundamentally changed the ways we consume and exchange information, as well as how we interact with one another. Driven by advances in microprocessor, storage, broadband network, and sensor technologies, more and more devices that capture areas of daily life are being connected to the Internet. Building on this trend of connected everyday objects and environments, IoT refers to a paradigm in which not only human-to-human or human-to-machine communication takes place over the Internet, i.e. in which machine-to-machine communication over the Internet is ubiquitous. The chapter *Internet of Things* briefly outlines the historical background of IoT, describes its enabling technologies and basic concepts, and delineates common usage cases of IoT.

Critical Information Infrastructures

Distributed IS have evolved rapidly in the past decades and have taken an increasingly central role in society. Today, some IS have become so critical to parts of society that their disruption or unintended consequences can have detrimental effects on vital societal functions, that is, they have become critical information infrastructures. The chapter *Critical Information Infrastructures* introduces the concept, distinguishes it from conventional critical infrastructures (e.g., electric power grids), outlines its main characteristics, and discusses important challenges, main functions, and core tasks for its operation.

1.3 Distributed Information Systems for Internet Computing

1.3.1 *Distributed Systems*

Owing to the Internet's inherent distributed nature, almost all Internet-based applications are distributed systems. Thus, it is important to understand what these are and how their characteristics influence the ways we should think about the design of Internet-based applications. In their highly influential book on distributed system, Andrew S. Tanenbaum and Maarten van Steen defined distributed systems as “a collection of independent computers that appears to its users as a single coherent system.” (Tanenbaum and Van Steen 2007).

Distributed System

A distributed system is a collection of independent computers that appears to its users to be a single coherent system (Tanenbaum and Van Steen 2007).

The definition has two key aspects: (1) a distributed system consists of components that are autonomous and (2) its users (people or other systems) think they're dealing with a single system. Thus, developers of distributed systems need to find ways to establish collaboration between the components of the system while hiding these interactions from users. The very well-known Leslie Lamport once defined a distributed system as “one in which the failure of a computer you didn't even know existed can render your own computer unusable.” (Buschmann et al. 2007). Developers of distributed systems must often ensure that they are continuously available, although some components may be temporarily out of order as they are replaced or fixed. One can generally say that the Internet provides the ideal basic infrastructure for such systems. However, utilizing unified network protocols, a global network of satellites, and glass fiber for data transmission is not sufficient to build distributed Internet-based applications. Developers of such systems may also face incompatibilities owing to heterogeneous operating systems and communication interfaces. In this book, I provide insights into some of the technologies, architectural paradigms, and systemic matters that can be utilized to overcome these challenges while designing and building distributed Internet-based applications.

1.3.2 *Information Systems*

When we reflect on distributed Internet-based applications in organizational contexts, we must introduce the concept of IS. Technological advancements in the past decades have fundamentally changed the way of how firms use and relate to technology. In particular, emerging information technologies have transformed how firms create value for customers by enabling the development of IS that

facilitate activities across businesses and industries. Thus, during the past decades, a substantial research community has formed around this topic. Kenneth C. Laudon and Jane P. Laudon defined IS as a set of interrelated components that collect (or retrieve), process, store, and distribute information to support decision-making and control in an organization. Further, IS may also help managers and workers to analyze problems, visualize complex subjects, and create new projects.

Information Systems

Information systems are interrelated components that work together to collect, process, store, and disseminate information to support decision-making, coordination, control, analysis, and visualization in an organization (Laudon and Laudon 1999).

A fundamental characteristic of IS is their sociotechnical nature (Orlikowski 1992). This means that IS not only comprise software, hardware, and data components. Instead, also organizational structures, the people that use such systems, and the interactions between the IS and its users in a social context (e.g., the organization) are seen as equally important and should therefore be considered when designing IS. Thus, IS researchers build on and combine knowledge from different related scientific fields such as computer science, engineering, psychology, business administration, and economics. Very often (but not always), Internet-based applications are distributed sociotechnical systems that serve specific organizational purposes. Thus, the scientific knowledge generated by IS researchers is crucial for Internet-based applications. Many of the thoughts on technologies, architectures, and systemic matters presented in this book have been fundamentally shaped by IS researchers and practitioners.

1.3.3 Design Challenges of Distributed Information Systems

Developers of distributed IS face various challenges when designing and implementing such systems. These challenges manifest differently, since every distributed IS has unique characteristics and is built on unique preconditions. However, core design challenges hold for all these systems and should be considered when they are designed, implemented, and operated. In this book, six core design challenges are introduced: (1) reliability, (2) scalability, (3) privacy and security, (4) integration, (5) interoperability, and (6) usability. It is important to keep these design challenges or quality attributes in mind while thinking about Internet-based applications and the ways they are designed. If one understands what these design challenges mean and why they are important, it is often much easier to understand the reasons why technologies, architectures, and systemic matters of Internet-based applications are designed the ways they are. The design challenges are explained in some detail below.

Reliability

Here, reliability is a system's ability to satisfactorily perform the task for which it was designed or intended, for a specified time and in a specified environment (Weik 2012). Software or hardware errors in Internet-based applications can lead to transient or persistent failures. While transient failures may be solved by restarting or replacing failed components, persistent failures involve unrecoverable data loss. Such failures can lead to high costs for the organizations that provide such systems. Costs may manifest for instance in the form of opportunity costs owing to missed transactions or claims for compensation by customers affected by such failures. Thus, developers must ensure that Internet-based applications work reliably.

Scalability

Popular Internet-based applications now handle tens of thousands of requests per second. However, the demand for such requests is not consistent over time. Instead, organizations that operate such Internet-based applications must be able to react and scale up their computing power swiftly in order to meet the growing demand and to maintain the system's availability and proper functioning. In this context, scalability can be defined as a system's ability to accommodate an increasing number of elements or objects in order to handle growing volumes of work, and/or to be susceptible to enlargement (Bondi 2000). Designing a suitable system architecture and choosing a suitable technological foundation are key to ensuring a system's scalability.

Information Security

Internet-based applications often store highly sensitive information (e.g., credit card numbers, health information, financial statements, or business contracts). Whether it is personal information about individuals or confidential business information, such data must be protected from unauthorized access and destructive forces. Information security is commonly divided into three main components: (1) confidentiality, (2) integrity, and (3) availability. Confidentiality refers to the ability to ensure that information is not made available or disclosed to unauthorized individuals, entities, or processes. It is probably the most obvious aspect of information security but is also very often attacked, for instance to steal financial information. Integrity is the ability to protect data from modification or deletion by unauthorized parties, and ensuring that when authorized people make changes that should not have been made, the damage can be undone. Availability refers to a system's ability to ensure that the information in question is readily accessible to authorized viewers at all time. Authentication mechanisms, access channels, and systems must all work properly so as to ensure the information is available when needed. Availability can be threatened by Denial-of-Service (DOS) attacks, which seek to temporarily shut down a system.

Integration

Developers of Internet-based applications have a wide variety of tools, such as programming languages (e.g., C++, Java, Python, Perl, JavaScript), database technologies (e.g., MySQL, MongoDB, PostgreSQL), and operating systems. Concerning the abovementioned design challenges (e.g., interoperability and

scalability), it is often beneficial to integrate loosely coupled systems instead of building single monolithic solutions. Popular Internet-based applications almost always integrate a mix of different technologies and components. Thus, developers must find ways for these technologies and components to communicate and interact with one another. In general, (system) integration can be defined as the process of linking different components physically or functionally in order for the system to be able to act as a coordinated whole.

Interoperability

Customers often access Internet-based applications by different means, including for instance, personal computers, mobile devices, or workstation clients, all characterized by different interfaces, screen sizes, operating systems, etc. Thus, Internet-based applications need to support heterogeneous clients. Further, organizations often need to automatically exchange data with other businesses' systems or services. Thus, organizations often need to integrate these services from other businesses (e.g., by utilizing Web services) or need to build proprietary interfaces for business-to-business (B2B) relationships based on individual agreements with partner firms. Overall, these requirements can be summarized under interoperability, which refers to the capability of two or more networks, systems, devices, applications, or components to exchange and readily use information securely, effectively, and with little or no inconvenience to users (NIST 2010).

Usability

Usability is a key factor that determines whether a system is accepted and used by its potential users (Davis et al. 1989). If users have difficulties using a system, they are likely to avoid it and are also potentially more inclined to search for alternatives. Usability is popularly conceptualized along three primary aspects: (1) effectiveness, (2) efficiency, and (3) satisfaction. Effectiveness refers to "the ability of users to complete tasks using the system, and the quality of the output of those tasks." (Brooke 1996). Efficiency is the relationship between effectiveness and the amount of effort needed (Schmidt-Kraepelin et al. 2014). Satisfaction refers to users' "subjective reactions to using the system." (Brooke 1996). Developers of Internet-based applications have a wide range of different tools to achieve sufficient usability (e.g., standardized questionnaires, eye-tracking, focus groups, and user testing). Whichever tools developers use, to reach high usability, developers must understand what people actually do with the system and whether this is in accordance with the developer's pre-understanding of how the system should be used to unfold its full potential.

1.4 Application Examples of Internet Computing

Originally, commercial use of ARPANET and also of the subsequent NSFNET were strictly prohibited. These networks were initially designed and used only for communication, research, and military purposes. Even today, many Internet-based applications are still proving the Internet's enormous potential for non-commercial purposes. One of the best-known examples of this is the free online encyclopedia

Wikipedia. However, since the Scientific and Advanced-Technology Act, 42 U.S.C. §1862(g) was passed by the U.S. Congress in 1992, which opened the doors for commercial use of the Internet, various different stakeholders have been seeking to use the Internet for commercial purposes, mainly by offering or utilizing Internet-based applications. As noted, Internet-based applications are highly heterogeneous. They come in various different forms, serve different target groups, and are based on fundamentally different technological foundations. This chapter illustrates this heterogeneity by introducing five of the most popular Internet-based applications and briefly describing how they differ in their architectural and technological structures.

Amazon

Buying or selling products or services over the Internet is one of the oldest and most traditional forms of the Internet's commercial use. These activities are generally referred to as e-commerce. The history of e-commerce dates back to the very beginning of ARPANET. In his 2005 book *What the Dormouse Said*, John Markoff described SRI and MIT students using ARPANET in the early 1970s to arrange cannabis sales, which he refers to as the seminal act of e-commerce (Markoff 2005). Amazon is now the largest e-commerce platform worldwide, with more than 300 million active customer accounts and USD 232.8 billion in annual revenue in 2018 (Amazon 2019). For certain special events or holidays, Amazon needs to provide a highly scalable system, since it must timeously deal with a vast amount of data. For instance, in 2016, Amazon Prime Day saw more than 600 orders per second (Popomaronis 2016). To handle these enormous amounts of data and maintain the service of buying goods online at all times, Amazon needs large computing resources, a sophisticated distributed system architecture, and a suitable selection of backend technologies. Owing to its need for highly flexible and available services, Amazon in 2006 founded the cloud computing service Amazon Web Services (AWS). However, instead of using it only for its own internal purposes, Amazon decided to capitalize on its newly developed resources and also to offer AWS to other organizations. Today, AWS is among the cloud computing companies with the highest annual revenue in the B2B market (Dignan 2018).

Dropbox

In the Business-to-Consumer (B2C) sector, Dropbox is among the most popular Cloud storage and file sharing services. According to own statements, Dropbox has more than 500 million active users (of which only 13.2 million are paying users) across 180 countries (Dropbox 2019a). Thus, Dropbox must deal with vast amounts of data from various sources. Billions of files are uploaded to Dropbox every day. To handle this, Dropbox was built based on Amazon Simple Storage Service (Amazon S3), a service provided by AWS. However, in 2016, Dropbox decided to move most of its infrastructure from Amazon S3 to an in-house solution called Magic Pocket (Gupta 2016). When Dropbox made this transition, it had to move 500 petabytes (i.e., five followed by 17 zeros) from AWS servers to its own data centers (Gupta 2016). As a file storage service, Dropbox pays particular attention to data security. Thus, all data stored on Dropbox servers are encrypted using Advanced Encryption Standard AES-256 encryption (Dropbox 2019b).

SAP ERP

SAP ERP is an enterprise resource planning software and the German company SAP SE's main product. It incorporates the key business functions of organizations, including sales, accounting, supply chain management, production management, payroll, and recruiting. With the launch of SAP R/3 in 1992, SAP ERP's architecture was based on a three-tier client server architecture (for more details on client server architectures, see Chapter 2) consisting of database server, application server, and client-sided presentation layer. In 2015, the newest version SAP S4/HANA was officially launched by SAP as the successor to SAP R/3 and SAP ERP. It is available either as an on-premise or as a cloud-based solution. However, moving organizations' ERP systems to the cloud is a difficult and costly undertaking. By 2018, SAP reported that about 20% of its annual revenue of €24.7 billion was a result of cloud subscriptions and support.

Bitcoin

Bitcoin has become the best-known and most widely used cryptocurrency. Bitcoin was originally invented by an unknown person or group of persons using the pseudonym Satoshi Nakamoto. The concept was first introduced in the 2008 paper *Bitcoin: A Peer-to-Peer Electronic Cash System* (Nakamoto 2008). The software code was released as open-source in January 2009. Bitcoin is based on distributed ledger technology. Transactions are verified by network nodes through cryptography and are recorded in a public distributed ledger called a blockchain. Its most decisive advantage over traditional currencies is that no bank or other verifying intermediary is needed to ensure reliable and secure transactions. Bitcoin's price development is characterized by high volatility. After reaching its all-time high of USD 19,783.06 on December 17, 2017 it dropped to USD 3,232.51 on December 15, 2018 (BTC Echo 2019). The reasons for this high volatility include increasing legal regulation and trading bans, especially in Asian countries such as China and South Korea.

Facebook

With around 2.375 billion monthly active users, as at 2019, [Facebook.com](https://www.facebook.com) is the most popular social media website (Facebook 2019b). Facebook was founded in 2004 by Mark Zuckerberg along with fellow Harvard students. [Facebook.com](https://www.facebook.com) (and its predecessor, [thefacebook.com](https://www.thefacebook.com)) originally restricted its membership to Harvard students. Gradually, students from other U.S. universities were added, until the network was finally opened up to everyone in 2006. Facebook's revenue model is predominantly based on advertising. In 2018, more than 98% of its annual revenues stemmed from advertising (Facebook 2019a). However, over the past few years, Facebook has been pursuing the strategy of gaining a foothold in other business areas through targeted acquisitions of other companies. These include instant messaging (the acquisition of WhatsApp), virtual reality (the acquisition of Oculus VR), and AI (the acquisition of Bloomsbury AI). Facebook's server-sided architecture was built on the LAMP Server stack. LAMP is an abbreviation of its four cornerstones: (1) Linux (operating system), (2) Apache (HTTP server), (3) MySQL (relational database management system), and (4) PHP (programming language).

Summary

The Internet has become an integral part of our private, business, and societal lives. Given its impacts and its omnipresence in our daily lives, it is all the more astonishing when one considers that the Internet as we know it has not existed for very long. In the course of its history, it has undergone three key phases: (1) development of the technological fundamentals, from the mid-1960s, (2) growth and internationalization, from the mid-1970s, and (3) commercialization, from the early 1990s. Although its predecessor, ARPANET, was originally a national project limited to only research and communication purposes and that explicitly prohibited commercial use, the Internet has led to the emergence of a variety of new business models that would have been impossible without it. This book examines the meanings of the Internet for organizations and businesses through the scientific field of Internet computing, which is composed of (1) applications (i.e. Internet-based applications provided via the Internet), (2) architectures (i.e. the basic components and structures that allow Internet-based applications to work), (3) technologies (i.e. the underlying technical capabilities that enable the development of such applications), and (4) systemic matters (i.e. important trends and paradigms that inform the design of Internet-based applications). Owing to the Internet's inherent distributed nature, almost all Internet-based applications are distributed systems. Thus, looking at knowledge developed in the field of distributed systems helps us to understand how the distributed nature of Internet-based applications influence how we should think about their design. Further, the scientific field of IS helps us to understand the meanings of aspects that go beyond pure technical deliberations. A key characteristic of any IS is its sociotechnical nature. This means that IS researchers and practitioners not only examine software, hardware, and data components; they also shed light on organizational structures, the people that use such systems, and the interactions between an IS and its users in a social context (e.g., an organization). They consider such aspects as equally important and examine how they shape the design of IS. This perspective helps us to understand the importance of such contextual aspects within Internet-based applications and how they influence how we should think about building Internet-based applications. When designing, implementing, and operating Internet-based applications, developers face core design challenges that are known from distributed IS; in this chapter, six were briefly introduced: (1) reliability, (2) scalability, (3) privacy and security, (4) integration, (5) interoperability, and (6) usability. At least since the U.S. Congress passed the Scientific and Advanced-Technology Act, 42 U.S.C. §1862(g) in 1992, various different stakeholders have been using the Internet for commercial purposes, mainly by offering or utilizing Internet-based applications, which are highly heterogeneous. For instance, they serve different target groups and are based on fundamentally different technological foundations.

Questions

1. What role did ARPANET play in the creation of the Internet?
2. How is Internet computing defined?
3. What are the key characteristics of distributed IS?
4. Why are distributed IS important for Internet-based applications?
5. What are core design challenges of distributed IS?
6. What are examples of Internet-based applications?

References

- Amazon (2019) Amazon annual report 2018. <https://ir.aboutamazon.com/annual-reports>. Accessed 15 Sept 2019
- Baran P (1964) On distributed communications networks. *IEEE Trans Commun Syst* 12(1):1–9
- Bauer JM, Latzer M (2016) Handbook on the economics of the internet. Edward Elgar, Cheltenham
- Beranek B, Newman (1981) A history of the ARPANET: the first decade. National Technical Information Services, Arlington County, VA
- Bondi B (2000) Characteristics of scalability and their impact on performance. Paper presented at the international workshop on software and performance, Ottawa, ON, 17–20 Sept 2000
- Brooke J (1996) SUS-A quick and dirty usability scale. *Usability Eval Ind* 189(194):4–7
- BTC Echo (2019) Bitcoin-Kurs. <https://www.btc-echo.de/kurs/bitcoin/>. Accessed 29 May 2019
- Buschmann F, Henney K, Schmidt DC (2007) A pattern language for distributed computing. In: *Pattern-oriented software architecture*, vol 4. Wiley, Chichester
- Cole J, Berens B, Suman M, Schramm P, Zhou L (2018) The 2018 digital future report – surveying the digital future. <https://www.digitalcenter.org/wp-content/uploads/2018/12/2018-Digital-Future-Report.pdf>. Accessed 15 Sept 2019
- Crocker S (1969) RFC 001: host software. Internet engineering task force. <https://tools.ietf.org/html/rfc1>. Accessed 20 Aug 2019
- Davis FD, Bagozzi RP, Warshaw PR (1989) User acceptance of computer technology: a comparison of two theoretical models. *Manag Sci* 35(8):982–1003
- Dignan L (2018) Top cloud providers 2018: how AWS, Microsoft, Google, IBM, Oracle, Alibaba stack up. *ZDNet*, 11 Dec 2018
- DiNucci D (1999) Fragmented future. *Print* 53(4):32–33
- Dropbox (2019a) Dropbox announces fiscal 2019 first quarter results. <https://dropbox.gcs-web.com/news-releases/news-release-details/dropbox-announces-fiscal-2019-first-quarter-results>. Accessed 29 May 2019
- Dropbox (2019b) Under the hood: architecture overview. <https://www.dropbox.com/business/trust/security/architecture>. Accessed 29 May 2019
- Facebook (2019a) Facebook annual report 2018. https://s21.q4cdn.com/399680738/files/doc_financials/annual_reports/2018-Annual-Report.pdf. Accessed 29 May 2019
- Facebook (2019b) Facebook Q1 2019 results. https://s21.q4cdn.com/399680738/files/doc_financials/2019/Q1/Q1-2019-Earnings-Presentation.pdf. Accessed 29 May 2019
- Ferré F (1988) *Philosophy of technology*. Prentice Hall, Englewood Cliffs, NJ
- Filman RE (2005) From the editor in chief: internet computing. *IEEE Internet Comput* 9(6):4–5
- Gupta A (2016) Scaling to exabytes and beyond. <https://blogs.dropbox.com/tech/2016/03/magic-pocket-infrastructure/>. Accessed 29 May 2019
- Kleinrock L (1961) Information flow in large communication nets. <https://www.lk.cs.ucla.edu/data/files/Kleinrock/Information%20Flow%20in%20Large%20Communication%20Nets.pdf>. Accessed 15 Sept 2019
- Kleinrock L (2007) *Communication nets: stochastic message flow and delay*. Dover Publications, Mineola, NY

- Kleinrock L (2010) An early history of the internet [History of communications]. *IEEE Commun Mag* 48(8):26–36
- Laudon KC, Laudon JP (1999) *Management information systems*, 6th edn. Prentice Hall PTR, Upper Saddle River, NJ
- Leiner BM, Cerf VG, Clark DD, Kahn RE, Kleinrock L, Lynch DC, Postel J, Roberts LG, Wolff S (2009) A brief history of the internet. *ACM SIGCOMM Comput Commun Rev* 39(5):22–31
- Licklider JCR, Clark WE (1962) On-line man-computer communication. Paper presented at the spring joint computer conference, San Francisco, CA, 1–3 May 1962
- Lins S, Thiebes S, Schneider S, Sunyaev A (2015) What is really going on at your cloud service provider? Creating trustworthy certifications by continuous auditing. Paper presented at the 48th Hawaii international conference on system sciences, Kauai, Hawaii, 5–8 Jan 2015
- Lins S, Grochol P, Schneider S, Sunyaev A (2016a) Dynamic certification of cloud services: trust, but verify! *IEEE Secur Priv* 14(2):66–71
- Lins S, Schneider S, Sunyaev A (2016b) Trust is good, control is better: creating secure clouds by continuous auditing. *IEEE Trans Cloud Comput* 6(3):890–903
- Markoff J (2005) What the dormouse said: how the sixties counterculture shaped the personal computer industry. Penguin Group, New York, NY
- Mesthene EG (1970) *Technological change: its impact on man and society*. Harvard University Press, Cambridge, MA
- Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>. Accessed 4 Sept 2019
- NIST (2010) NIST framework and roadmap for smart grid interoperability standards, Release 1.0. https://www.nist.gov/sites/default/files/documents/public_affairs/releases/smartgrid_interoperability_final.pdf. Accessed 15 Sept 2019
- Orlikowski WJ (1992) The duality of technology: rethinking the concept of technology in organizations. *Organ Sci* 3(3):398–427
- Popomaronis T (2016) Prime day gives Amazon over 600 reasons per second to celebrate. 13 July 2016
- Roberts LG (1967) Multiple computer networks and intercomputer communication. Paper presented at the 1st ACM symposium on operating system principles, Gatlinburg, TN, 1–4 Oct 1967
- Schmidt-Kraepelin M, Dehling T, Sunyaev A (2014) Usability of patient-centered health it: mixed-methods usability study of EPILL. Paper presented at the eHealth, Athens, 12–14 May 2014
- Singh MP (2004) *The practical handbook of internet computing*, Chapman & Hall/CRC Computer and Information Science Series. CRC Press, Boca Raton, FL
- Speed T, Ellis J, Korper S (2001) *The personal internet security guidebook: keeping hackers and crackers out of your home*. Academic Press, San Diego, CA
- Tanenbaum AS, Van Steen M (2007) *Distributed systems: principles and paradigms*, 2nd edn. Prentice-Hall, Upper Saddle River, NJ
- Thierer A (2014) Defining “Technology”. <https://techliberation.com/2014/04/29/defining-technology/>. Accessed 3 Apr 2019
- Weik M (2012) *Communications standard dictionary*, 3rd edn. Chapman & Hall, New York, NY

Further Reading

- Laudon KC, Laudon JP (1999) *Management information systems*, 6th edn. Prentice Hall PTR, Upper Saddle River, NJ
- Leiner BM, Cerf VG, Clark DD, Kahn RE, Kleinrock L, Lynch DC, Postel J, Roberts LG, Wolff S (2009) A brief history of the internet. *ACM SIGCOMM Comput Commun Rev* 39(5):22–31
- Tanenbaum AS, Van Steen M (2007) *Distributed systems: principles and paradigms*, 2nd edn. Prentice-Hall, Upper Saddle River, NJ

Chapter 2

Information Systems Architecture



Abstract

Information systems (IS) are composed of different, interrelated elements that aim to fulfill desired features. This chapter introduces the concept of IS architecture as a way to meaningfully describe and design the underlying structure of an IS. It presents nine basic principles of IS architectures and explains them using the example of the cloud-based storage service Dropbox. This introduction to IS architecture also introduces the most common architectural patterns (i.e. the client-server architecture, tier architectures, peer-to-peer architecture, model view controller, and service-oriented architecture) in the realm of Internet computing and briefly discusses their strengths and weaknesses.

Learning Objectives of this Chapter

The main learning objective of this chapter is to understand the concept of IS architecture and its roles in the development of distributed IS in the realm of Internet computing. Having read this chapter, readers will understand the purposes of IS architectures and why architectural models can look different although they represent the same system. By getting to know the basic principles of IS architecture, readers will be able to understand how properties of IS architectures can influence system features and behaviors. Finally, readers will get to know the most important and widespread architectural patterns.

Structure of this Chapter

This chapter is structured as follows. Section 1 introduces the concept of IS architecture by providing a definition and discussing its main purposes in the development of distributed IS. Section 2 introduces the nine principles of IS architecture. In the third section, the four fundamental architectural views (i.e. the logical view, the process view, the development view, and the physical view) proposed by Kruchten (1995) are outlined. The chapter concludes by introducing a selection of the most common and widespread architectural patterns (i.e. the client-server architecture, tier architectures, peer-to-peer architecture, model view controller, and service-oriented architecture).

2.1 Defining Information Systems Architecture

In an information system (IS) context, the term “architecture” has been used in several ways and at different levels of abstraction. For instance, enterprise architectures describe structures, business processes, and infrastructures within complex organizations (Lankhorst 2017). Hardware architectures focus on the technical components of often complex infrastructures, and software architectures concentrate on software systems of different sizes and different scopes. Traditionally, in a business context, software architectures were bound to software used in the context of focal organizations. With the rise of globally distributed networks and partnerships, inter-organizational IS originated that required more complex architectures and that were often distributed across companies, industries, and countries. Thus, today, IS architectural considerations are often not limited to organizational boundaries, but may cover all kinds of relationships to external entities.

In a more general sense, architecture refers to the foundational elements of a system, these elements’ interrelationships, and principles that govern the system’s design. A popular definition comes from the ISO/IEC/IEEE 42010 standard, which creates a common frame of reference for systems and software engineering practices. In the 2011 version of the standard, which was reconfirmed in 2017, a system’s architecture is defined as “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.” In this book, IS architecture describes the architectural foundations of applications, particularly in the realm of Internet computing.

Information System Architecture

Fundamental concepts or properties of an information system in its environment, as embodied in its elements and relationships, and in the principles of its design and evolution (ISO/IEC/IEEE 2011).

IS architecture considerations can involve multiple different views of the same IS by multiple stakeholders with fundamentally different needs (e.g., developers, system architects, enterprise architects, and business representatives). These views can be understood as projections of an IS based on these different actors’ perceptions (Golden 2013). Generally, each view is an analytical description of an IS. Considering multiple viewpoints allows one to compensate for the weaknesses of analytical decomposition within single views. Thus, one may say that the set of all of these are the best approach to define the overall IS architecture. However, it is generally impossible to define an IS in an objective, unified, and exhaustive way. Different architectural views can address different information needs of stakeholders; these include for instance software developers, who need to understand the software’s underlying architecture and its interfaces to adjacent systems, or IT and business managers, who require an overview of a firm’s application landscape in order to take investment decisions. Such a broad focus on architectures has been a

dominant view in research under the umbrella of enterprise architecture management (Winter and Fischer 2006; Boh and Yellin 2006).

To make the architecture of an IS tangible and to be able to communicate it to other stakeholders, IS architects use so-called architectural models (Zachman 1987). Architectural models may be referred to as illustrations, created using available standards, in which the primary concern is to represent an IS's architecture. Architectural models are representations of an IS's architecture from a specific perspective and for specific purposes. Architectural models should represent the IS under consideration in a way that can be understood by all involved stakeholders. The most common standard for architectural models is the graphic modeling language Unified Model Language (UML) (ISO/IEC 2005).

Architectural Model

An illustration, created using available standards, in which the primary concern is to represent the architecture of an IS from a specific perspective and for a specific purpose.

IS architectural models can serve various purposes. First, it is necessary for IS architects to think deeply about IS architecture in order to design a new IS that is able to meet its non-functional and functional requirements (Sommerville and Sawyer 1997) that have already been identified in a requirements engineering phase (Castro et al. 2002). On the other hand, architectural models can also serve as a tool to document the components and their interrelationships of an existing IS. To this end, architectural models can facilitate an understanding of an IS and its evolution. IS architectural models generally provide a common language for all stakeholders involved with an IS. They enable stakeholders to reason about an IS' structural properties. In particular, IS architectural models are important means to facilitate communication between stakeholders about complex dependencies between various physical and virtual components. Another purpose of reasoning about IS architectural models is that they may provide best practices and lessons learned that can be re-used at a larger scale or may be transferred to a different application context.

2.2 The Principles of Information System Architecture

Regardless of the IS type under consideration, certain principles can be applied to every IS architecture. It is important to keep these principles in mind, because they help one to understand why IS architecture models can look very different although they are representations of the same system. Consequently, in the following, the nine principles for system architecture proposed by Golden (2013) are introduced. They were slightly adjusted to ensure that they fit the context of IS in Internet computing. Overall, these principles can help us to understand that a good architectural approach

strongly depends on various factors, such as the target group of an IS architecture representation or the abstraction level. To make it easier for readers to understand these principles, I explain them using the example of the cloud computing service Dropbox.

2.2.1 *Principle 1: Architecture Models Information System Boundaries, Inputs, and Outputs*

The first principle of IS architecture states that an IS under consideration is modeled as a system that performs certain functions. First, the IS needs to be defined by its perimeters, which means that IS architects must clearly define the IS's boundaries. An IS may also be defined by certain inputs and outputs. Finally, the IS can have an internal state that determines which functions the IS performs, which inputs it captures, and which outputs it provides (Fig. 2.1).



Fig. 2.1 Architecture models information system boundaries, inputs, and outputs

Example: The cloud computing service Dropbox is an IS that takes certain inputs such as data files and user configurations and provides the ability to access files, from anywhere, from every device, and provides customized access to files stored in Dropbox as output. The service's functions includes data access from multiple devices that are connected to the Internet or data sharing with selected others.

2.2.2 *Principle 2: An Information System Can Be Broken down into a Set of Smaller Subsystems*

The second principle states that every IS can be broken down into a set of interconnected subsystems. The IS under consideration would be the sum of all these subsystems. In turn, subsystems could again be the subject of an architectural examination and can therefore potentially be broken down into another set of subsystems on a second level that, together, form a first-level subsystem. However, notably, IS are mostly very complex. An IS cannot be fully understood by its entire set of subsystems. Instead, relationships between the different parts (i.e. subsystems) of a complex IS give rise to collective behavior by this IS as a whole (Fig. 2.2).

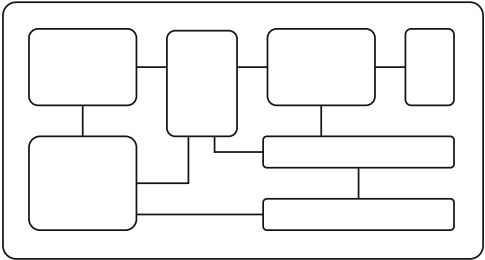


Fig. 2.2 An information system can be broken down into a set of smaller subsystems

Example: Dropbox can be divided into several subsystems, such as an authentication system that ensures that only authorized users can access certain data files, an encryption subsystem that is responsible for encrypting data before it is stored, and a storage server subsystem that ensures that data is stored reliably (Dropbox 2019). However, notably, considering solely the sum of these subsystems does not comprehensively explain Dropbox’s overall behavior. Instead, IS architects must also understand how these subsystems interact with one another (see principle 5).

2.2.3 Principle 3: An Information System Can Be Considered in Interaction with Other Systems

No IS exists in a void. Instead, IS interact with various other IS in some way. The third principle states that every IS can be considered in interaction with other IS. An IS may for instance use data gathered by another IS or may provide functionalities for another IS. While principle 2 describes the decomposition of systems into smaller subsystems, principle 3 states that individual IS, taken together, may form higher-level IS. However, IS architects are primarily interested in how a specific IS connects to other IS in its immediate environment (e.g., which inputs it receives from other systems and which outputs it must provide to other systems) and often do not necessarily need to fully understand the internal functions of related IS (Fig. 2.3).

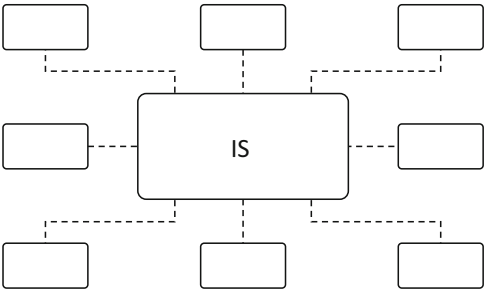


Fig. 2.3 An information system must be considered in interaction with other systems

Example: Dropbox interacts with various systems utilized simultaneously by end-users. Thereby, the end-user systems themselves are highly heterogeneous, since they utilize different operating systems, devices, screen sizes, etc. It is the task of the responsible IS architects to recognize and account for these different requirements of heterogeneous user systems on Dropbox’s system architecture.

2.2.4 Principle 4: An Information System Can Be Considered Through Its Entire Lifecycle

The fourth principle states that IS architects can consider an IS through its entire lifecycle. An IS typically goes through various stages during its lifetime. This includes the design phase, the development phase, the test phase, and the operation phase. Depending on which phase is of interest, different requirements can be placed on the architecture (Fig. 2.4).

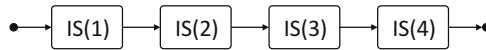


Fig. 2.4 An information system can be considered through its entire lifecycle

Example: The architecture of cloud computing services such as Dropbox change continually as it seeks to meet new functional and non-functional requirements (Lins et al. 2015; Lins et al. 2016b; Lins et al. 2016a). For instance, when Dropbox decided to move its infrastructure from Amazon Simple Storage Service (Amazon S3) to an in-house solution called Magic Pocket (see Chapter 1), this necessitated far-reaching changes to its architecture.

2.2.5 Principle 5: An Information System Can Be Linked to Another Information System via an Interface

As stated by principle 3, every IS is connected in some way to other IS, and no IS exists in isolation. Principle 5 states that these connections between two IS can be modeled via interfaces that explain the mechanisms of how the two IS interact (Fig. 2.5).

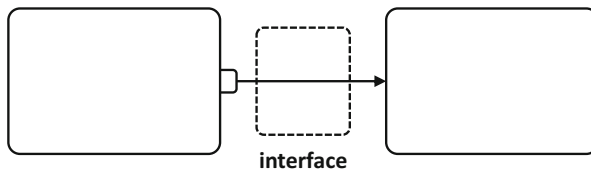


Fig. 2.5 An information system can be linked to another information system via an interface

Example: Users have various highly heterogeneous requirements when accessing the services provided by Dropbox. Thus, Dropbox must enable users to access its services in a convenient, reliable, and secure way via interfaces. For most private users, Dropbox does this by providing client applications for the most common operating systems, such as Microsoft Windows, Mac OS, iOS, and Android, besides access through a Web browser. However, especially commercial users of Dropbox may have additional requirements. Thus, Dropbox provides the opportunity to access its services via an application programming interface (API) called DBX Platform in order to enable more flexible access to its services. DBX Platform also helps to disseminate Dropbox by allowing developers to integrate Dropbox services into the architecture of their own applications.

2.2.6 Principle 6: An Information System Can Be Modeled at Various Abstraction Levels

As stated by principle 2, an IS can be broken down into a set of subsystems, but the sum of the subsystems does not sufficiently describe the IS's characteristics. Principle 6 states that an IS can be subject to architectural considerations at different abstraction levels. The appropriate abstraction level strongly depends on the IS's properties and behaviors, which are of interest to the architect. For instance, an IS architect, who is only interested in the generated output of the IS (e.g., because they solely want to use its API functionality), will likely choose a very high abstraction level. On the other hand, an architect who wants to alter certain functionalities of the IS likely needs to conduct architectural considerations at a lower abstraction level in order to know how a change in functionality affects the IS's other components (Fig. 2.6).

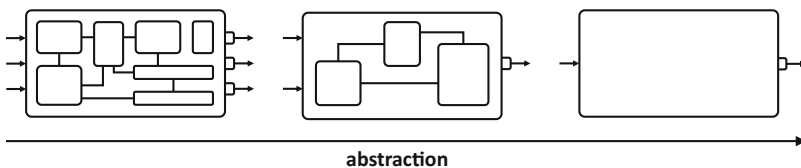


Fig. 2.6 An information system can be modeled at various abstraction levels

Example: Dropbox's architecture can be considered at various different abstraction levels. It can for instance be considered as a holistic cloud computing service to store and share data files, as a set of interconnected hardware devices, or as a set of software code. All these different abstraction levels are realistic and valid, but their relevance strongly depends on the architecture modeling's purpose.

2.2.7 Principle 7: An Information System Can Be Viewed Along Several Layers

Principle 7 states that an IS architecture can be viewed along several layers. The main benefit of dividing an IS into layers is that this enables stakeholders in the IS to reason in an isolated way about specific aspects contained in one layer. There are various approaches to how these layers are constructed and based on one another. According to Golden (2013), it is always possible to propose three layers and describe them in conjunction with three questions: the system's purpose (why?), its functions (what?), and its construction (how?). In IS architecture, the most common approach is to divide an IS into logical layers that describe a grouping of the its functions (see Chapter 2.4.2) (Fig. 2.7).

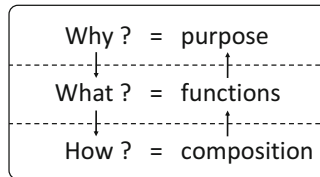


Fig. 2.7 An information system can be viewed along several layers

Example: Dropbox is a cloud computing service that seeks to enable users to store and receive data files. Yet it is also a set of functions that serve this purpose (e.g., upload and download data files, display users' available stored data files). Further, all these functions are implemented via physical (i.e. hardware) and non-physical (i.e. software) components.

2.2.8 Principle 8: An Information System Can Be Described Through Interrelated Models with Given Semantics

Principle 8 states that an IS can be described by a set of different models that are interrelated through given semantics. This principle refers to the fact that an IS's behavior is not static, but can depend on a variety of different aspects, such as user inputs or the IS's current state. For instance, a firm's IS may shut down certain functionalities during a weekend owing to maintenance. Accordingly, the IS must be designed in such a way that it performs only a reduced set of functionalities and potentially displays errors messages when receiving user inputs that seek to trigger unavailable functionalities (Fig. 2.8).

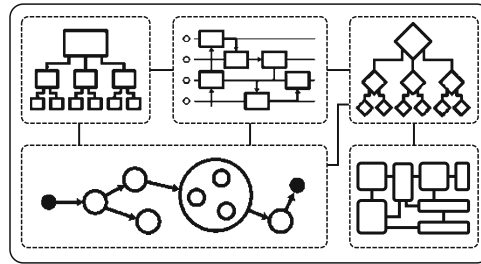


Fig. 2.8 An information system can be described through interrelated models with given semantics

Example: Dropbox offers its users the possibility to share data with people that are not registered as users. Thus, an example of the manifestation of this principle in Dropbox is the fact that the IS must treat users differently depending on whether they are logged in or just received permission to access data (e.g., via a customized access link). This is particularly important for Dropbox to notice, since it provides different functionalities for these different user types.

2.2.9 Principle 9: An Information System Can Be Described Through Different Perspectives

An IS can be described through various perspectives that potentially correspond to different stakeholders who interact with the IS. All these stakeholders have fundamentally different notions of the IS and thus have different requirements on how to model its architecture. Principle 9 states that an IS can be described through various perspectives, which lead to heterogeneous perceptions of an IS (Fig. 2.9).

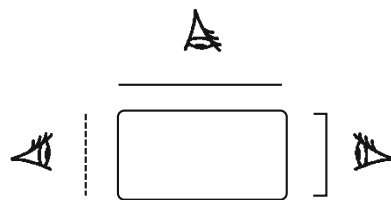


Fig. 2.9 An information system can be described through different perspectives

Example: A private user of Dropbox is probably most interested in which functionality the service offers them and is therefore not interested in technical details or the underlying hardware components of Dropbox. Other stakeholders, such as the organization that supplies Dropbox with server hardware, probably

need a much more detailed model of Dropbox's technical architecture in order to decide how new hardware may be integrated into the system.

2.3 Architectural Views

Each architectural model is a representational abstraction of an IS's architecture that fits a particular purpose. Thus, it is impossible to capture all the information relevant to all potential stakeholders in a single model. Instead, in most cases, it is advisable to create multiple models of an IS's architecture from different perspectives. In research and practice, there are different opinions on how many views are necessary to sufficiently design or document an IS. For instance, Kruchten (1995) proposed his famous 4+1 architectural view model, which contains four fundamental architectural views that are interrelated via scenarios and use cases: (1) the logical view (object-oriented decomposition), (2) the process view (process decomposition), (3) the development view (subsystem decomposition), and (4) the physical view (mapping the software to the hardware). I will now explain these four views in some detail.

The Logical View

The logical view primarily supports the realization of functional requirements, i.e. what the IS should provide in terms of services to its users. For this purpose, the IS can be decomposed into a set of key abstractions that are taken primarily from the problem domain and are represented in the form of objects. This decomposition is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across the IS's various parts.

The Process View

The process view seeks to represent the dynamic aspects of an IS by explaining its runtime behavior and by shedding light on its processes and how these communicate with one another. In the process view, a process is defined as a group of tasks that form an executable unit. Processes and tasks may also be replicated to increase the processing load's distribution, or to increase availability. The process view addresses issues of concurrency and distribution, IS integrity, fault tolerance, and how the main abstractions from the logical view fit the process architecture. The IS's process architecture can be described at several levels of abstraction, each addressing different concerns.

The Development View

The development view focuses on how the software module is organized in the software development environment. In the development view, the IS's software is packaged in small chunks that can be developed by one or a small number of developers. The subsystems are organized in a hierarchy of layers, each providing a narrow and well-defined interface to the layers above it. For the most part, the

development architecture considers internal requirements relating to software management, code re-use or commonality, and the constraints imposed by the toolset, or the programming language. The development view serves as the basis for requirements allocation, for the allocation of work to teams (or even for team organization), for cost evaluation and planning, for monitoring the project progress, and for reasoning about code re-use, portability, and security.

The Physical View

The physical view depicts an IS from the perspective of an IS engineer. It is concerned with the topology of IS components on the physical layer as well as the physical interconnections between these components. Thus, it is used to ensure suitable mapping of the IS's software components to available hardware resources.

2.4 Architectural Patterns

In IS design (as in almost every form of software development), developers often encounter recurring problems. To avoid having to redesign an IS architecture from scratch every time one encounters such a problem, certain best practices have been created over time. In the context of software development and architectural design, these best practices are referred to as architectural patterns (Avgeriou and Zdun 2005). Various different architectural patterns are now widely used as ways to present, share, and re-use knowledge about the architecture of software systems. A major trigger for the dissemination of architectural patterns was the publication of the book *Design Patterns. Elements of Reusable Object-Oriented Software* in 1995 (Gamma et al. 1995), followed by a number of other publications on architectural patterns (e.g., Buschmann et al. 2007; Kircher and Jain 2013; Schmidt et al. 2013). An architectural pattern is an abstract description of a recommended architectural approach that has been tested and proven successful in different IS and environments. It abstractly describes a possibility to structure a IS's architecture in ways that were successful in previous IS. Architectural patterns should also contain information about the circumstances in which the pattern should be used and should elaborate on its strengths and weaknesses. Notably, although an architectural pattern conveys a representation of an IS, it is not an architecture per se. Instead, various different IS architectures may implement the same pattern and may share related characteristics.

Architectural Pattern

An architectural pattern is an abstract description of a recommended architectural approach that has been tested and proven in different information systems and environments.

In the following, this chapter presents various fundamental patterns that have been established in the design of Internet-based applications. Since it would be impossible to describe all the generic patterns used in the development of Internet-based applications. Therefore, descriptions are limited to selected relevant architectural patterns. These include client-server, tier, peer-to-peer, model view controller, and service-oriented architectures.

2.4.1 Client-Server Architectures

Probably the most fundamental and important concept in the architectural design of Internet-based applications is the client-server model. The basic idea of this model is to distribute tasks or workloads between the providers of a resource or service (servers) and service requesters (clients). The third component of a client-server architecture is the network used by clients and servers to communicate with one another. A server runs one or more programs that share the server's resources with clients. A client does not share any of its resources, but requests a server's content or service function. Thus, clients initiate communication sessions with servers, which await incoming requests. Whether a computer is a client, a server, or both is determined by the nature of the application that requires the service functions. For instance, a single computer can simultaneously run a Web server and a file server to serve different clients' data needs. Clients can also communicate with a server in the same computer.

The key advantage of the client-server architecture is its distributed nature. Typically, it is fairly easy to add and integrate new participants to the IS (new clients) or to upgrade servers without directly affecting the IS's other parts. However, when integrating a new server, changes to existing clients (e.g., the provision of the new server address and its services) may be necessary. A major drawback of client-server architectures is that every server constitutes a single point of failure that may become unavailable owing to a high number of client requests or owing to Denial-of-Service attacks.

IS based on client-server architectures have led to a various important advances in information technology. For instance, closely related to client-server architectures is the development of remote procedure call, a communication mechanism that allowed a client and a server to interact with each other (see Chapter 5). Further, to develop clients that are able to communicate with the server, the server needed to have a known, stable interface. This resulted in the development of APIs (Alonso et al. 2004).

In a client-server architecture, tasks, workloads, and capabilities can be distributed along the distributed IS's participants (clients and servers), to different degrees. For instance, if an IS contains so-called thin clients, the server does most of the work, which may include performing complex calculations and storing data. A thin client is a lightweight computer that is optimized to establish a remote connection to the network, capturing user input, and displaying output. Thin clients have the

advantage of making the client easier to port, install, and maintain. They also require lower processing capacity in the client machine and can therefore be used by a wider range of computers. In contrast, a fat client typically provides rich functionality independent of the servers and is often characterized by the ability to perform many functions without this connection. However, a fat client still requires at least periodic connection to the network and servers.

2.4.2 Tier Architectures

IS architects often decompose IS into layers that represent a logical grouping of functions. Layered architectures provide architects with the opportunity to create flexible and re-usable IS. The basic idea of layered architecture is that layers are decoupled as much as possible from one another so as to minimize dependencies between them. By segregating an IS into layers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire IS. In theory and in practice, layered architectures are often referred to as tier architectures (a term I also use in this book). However, notably, while layer and tier are often used interchangeably, a fairly common perspective is that there is a difference. In this view, a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the IS infrastructure. For instance, a three-layer solution could easily be deployed on a single tier, such as a personal workstation. The most common classification has three layers: (1) the presentation layer, (2) the application layer, and (3) the data management layer (Alonso et al. 2004). In this chapter, I use these three main layers to explain the most common tier architecture types. I will now explain the three layers introduced by Alonso et al. (2004) in some detail.

The Presentation Layer

Every Internet-based application communicates with other entities (e.g., persons, other IS) in some way. To enable this communication, information must be presented in some form to these external entities and must allow them to interact with the IS. The components in an IS that are concerned with these tasks form the presentation layer. Sometimes, authors refer to the presentation layer as the IS's client. However, as Alonso et al. (2004) state, this is not always correct. In fact, the client of an Internet-based application can be completely external and thus not part of the IS. A good example of such a design are systems accessed through Web browsers using plain Hypertext Markup Language (HTML) documents. In this case, the client is the Web browser (e.g., Mozilla Firefox), which requests a Web page from a Web server. It merely displays the information generated by the Web server by interpreting the HTML content retrieved in the Web server's response. However, the Web browser is not part of the IS. In this example, the IS's presentation layer would contain all the components involved in creating the HTML documents.

The Application Layer

IS typically do more than simply deliver and present some form of information and data to external entities. In most cases, they perform some sort of data processing in order to be able to deliver and present the desired results. This involves a program that implements the de facto operation necessary to fulfill the client's request through the presentation layer. All these programs and the components that help to deploy and run such programs are referred to as the application layer. A typical example would be a program that implements the operations necessary for a withdrawal request from a bank account. This program would for instance take the request, verify that the account contains sufficient funds, check whether or not the withdrawal limit is being exceeded, create a log entry, update the current account balance, and give the approval to hand over the money.

The Data Management Layer

IS almost always need some form of data with which they can work. This data can reside in data repositories such as databases or file systems. All the IS's components that contribute in some way to the ongoing storage of the necessary data are referred to as the data management layer. In the banking example, the data management layer could be the bank's account database. Notably, many architectures also include, as part of the data management layer, other external systems that provide information. This may include not only data repositories, but also external systems that are themselves made up of presentation, application, and data management layers. Thus, IS architects may design IS recursively by using other IS as components of the IS under consideration.

Generally, in a layered architecture, a layer may only send a request to a layer below it. There are two layering architecture types. In strict layering architectures, no layers may be skipped, and a layer is only allowed to request functionality from the layer immediately below. Thus, high flexibility is guaranteed since, in the worst case, only adjacent layers must be adapted when changing a layer. Other IS architecture layers are not affected by the adjustments. Thus, the individual layers are easier to test and to maintain. In loose layering architectures, a layer may call any other lower layer. An advantage is that loose layering systems are often more efficient than strict layering ones, since the data does not have to be passed through intermediate layers before it arrives at the required layer. However, this leads to higher interdependencies between layers, which is usually undesirable owing to higher efforts concerning changes in the IS's architecture. I will now introduce and explain the most common tier architecture types, as introduced by Alonso et al. (2004), along common application examples.

One-Tier Architectures

Historically, one-tier architectures can be understood as the direct result of the computer architectures used several decades ago. These were mainframe-based, and interaction with the IS took place through dumb terminals that merely displayed information previously prepared by the mainframe. During this time, IS architects' main concern was efficient use of the limited computing power. Thus, from a layered architecture perspective, the presentation, application, and data management layers

were merged into a single tier because architects had no other option, owing to limited computing power. Thus, users interacted with such IS via terminals (i.e. the IS's clients), which were barely more than keyboards and computer screens. In one-tier systems, the entire presentation layer resides within the server. It controls every aspect of the interaction with the client, including how information appears, how it is displayed, and how to react to input from the user. One-tier IS have some clear advantages. For instance, architects are free to merge the layers as much as necessary. This approach can be used to optimize performance, since interactions between different layers always involve computing effort and overhead for encapsulating functionality within a layer. Further, where portability is not an issue, a one-tier system can liberally use assembly code and low-level optimizations to increase throughput and to reduce response time. These characteristics can result in an extremely efficient high-performance IS. The drawback of one-tier architectures is that they are often monolithic pieces of code. Thus, they are hard and expensive to maintain. In some cases, it has become almost impossible to modify such an IS since there is a lack of architectural understanding, owing to insufficient documentation and a lack of qualified programmers capable of dealing with such IS. Although it would now be possible to develop a one-tier IS, the software industry has been moving in the opposite direction for many years. Thus, in most cases, one-tier architectures are today only relevant to those who must deal with old mainframe legacy systems. Fig. 2.10 illustrates a typical one-tier architecture that contains all three logical layers in one monolithic system.

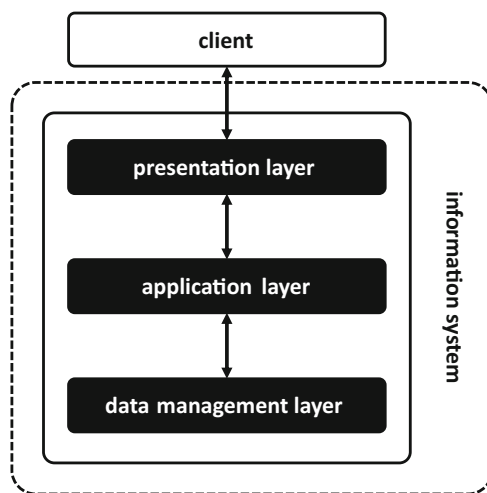


Fig. 2.10 A one-tier architecture (adapted from Alonso et al. (2004))

Two-Tier Architectures

With the emergence of the personal computer (PC), IS started to become more than a single one-tier mainframe. For IS architects, it was no longer necessary to keep the presentation layer together with the application and data management layer on a single machine. Instead, it was now possible to move the presentation layer to the IS's clients (i.e. to the PCs)—two-tier architectures were born. Two-tier architectures have two main advantages over one-tier architectures. First, utilizing PCs' computational power for the presentation layer resulted in more available resources for the application and data management layers on the mainframes and servers. Second, by separating the presentational layer, it was now possible to tailor it for different clients and purposes without increasing the IS's complexity. For instance, it was possible to build one presentation layer for administration purposes and another for ordinary end-users. Further, the different presentation layers were independent of one another and thus could be developed and maintained separately. Two-tier architectures have become enormously popular. In fact, when people today speak of client-server architectures (see Chapter 2.4.1), they most likely think of the classic two-tier architecture. In this case, the client typically contains the presentation layer and the client software, while the server encompasses the application and data management layers. Because of this, many people still identify the presentation layer with the client on which it runs. Two-tier architectures offer significant advantages over one-tier architectures. For instance, two-tier IS support development of IS that are portable across different platforms, since the presentation layer is independent of the other two layers. Accordingly, developers can provide customize presentation layers to different client types without making changes to the application and data management layers on the servers. A typical disadvantage of two-tier architectures is the legacy problem, which stems from wrong usage of the architecture. This arises when the client has integrated services from different servers; this is not a bad idea per se, but in order to do so, the client must be able to understand and utilize each server's API. Further, the client must combine the data from both servers, must deal with the exceptions and failures of both servers, and must coordinate the access to both servers. All these additional requirements make the client more complex and result in an additional application layer embedded in the client. Thus, in spite of their success, two-tier architectures have acquired the reputation of being less scalable and less flexible concerning integrating different IS. Fig. 2.11 shows a typical two-tier architecture where the application layer and the data management layer reside on the server, while the presentation layer is placed on the client.

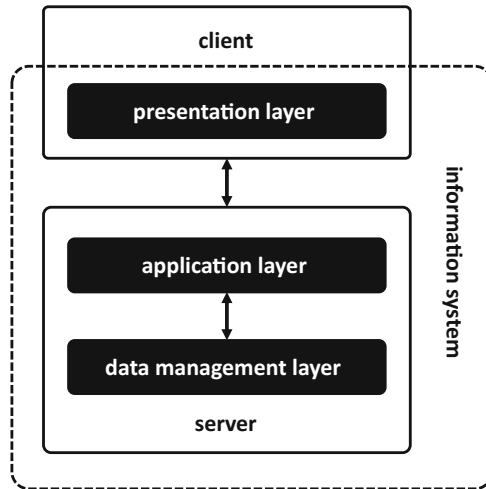


Fig. 2.11 A two-tier architecture (adapted from Alonso et al. (2004))

Three-Tier Architectures

Three-tier architectures seek to solve the problem of highly complex clients by introducing an additional tier between clients and servers. Three-tier architectures are far more complex and varied than one-tier or two-tier architectures. Three-tier architectures are usually based on a clear separation between each of the three layers. The presentation layer resides at the client, as in two-tier architectures. The application layer resides in the middle tier. For this reason, infrastructures that support the development of the application logic are collectively known as middleware (see Chapter 5). The data management layer is composed of all servers that the three-tier architecture seeks to integrate. The advantage of separating the application layer from the data management layer is that scalability can now be accomplished by running each layer on a different server. In particular, the application layer can be distributed across several nodes, so that it is even possible to use clusters of small computers for this layer. Further, three-tier IS offer the opportunity to write application logic that is less tied to the underlying data management and is therefore more portable and re-usable. The disadvantage is that the communication between the data management layer and the application layer requires much more effort and becomes more expensive in terms of computing power. In particular, if the IS only contains one server, a two-tier architecture is almost always more efficient than a three-tier architecture. However, the demand for application integration, flexible architectures, and portable application logic cannot be met with two-tier IS. As with two-tier architectures, the disadvantages of three-tier architectures are also due to a legacy problem. While two-tier IS run into trouble when clients want to connect to more than one server, three-tier IS run into trouble when the integration must happen via the Internet or involves different three-tier IS. Although three-tier IS and even

two-tier IS can be made to communicate via the Internet, most IS were just not designed for this purpose. Fig. 2.12 illustrates a three-tier architecture in which the application layer functionalities are performed by middleware.

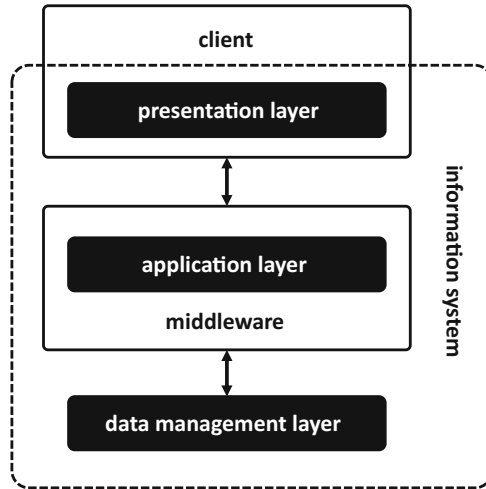


Fig. 2.12 A three-tier architecture (adapted from Alonso et al. (2004))

Multi-Tier Architectures

Multi-tier architectures were the result of the increased relevance of the Internet as an access channel. These architectures arise for instance from the need to incorporate Web servers as part of the presentation layer. In multi-tier architectures, the Web server is treated as an additional tier, since it is much more complex than most presentation layers. In such IS, the client is a Web browser and the presentation layer is distributed between the Web browser, the Web server, and the code that prepares HTML pages. Additional tiers may also be necessary, for instance, to translate between the different data formats used in each layer. As multi-tier architecture demonstrates, the architecture of most IS is now very complex and can encompass many different tiers as successive integration efforts lead to IS that later themselves become building blocks for further integration into other IS. In fact, the main disadvantage of the multi-tier model is that there is often middleware involved that contains redundant functionality. The efforts and costs of designing, developing, and maintaining these IS strongly increases with the number of tiers. Many multi-tier IS today encompass a large collection of networks, single computing devices, clusters, and links between different IS. Thus, in a multi-tier IS, it is sometimes hard to identify where one system ends and the next begins. Remote clients of all types access the system via the Internet. Their requests are forwarded to a cluster of machines that, together, comprise the Web server. Internally, there may be additional clients across the organization who also use the IS's services. It is also very common

to see the application logic distributed across a cluster of machines, since there may even be several middleware platforms for different applications and functionalities that coexist in the same IS. Underlying all this machinery, the back-end constitutes the data management layer.

2.4.3 Peer-to-Peer Architectures

Another architectural pattern is peer-to-peer. This is a commonly used architectural pattern in which every participant (a participant of the network is often referred to as peer or node) has the same capabilities and responsibilities (Fig. 2.13). In a pure peer-to-peer network, all peers have equal rights and can both use and provide services, in contrast to the traditional client-server architecture, in which the consumption and supply of resources is divided among clients and servers (Steinmetz and Wehrle 2004). The peers make a portion of their resources (e.g., processing power, storage, or network bandwidth) directly available to other network participants without the need for central coordination by servers or stable hosts. While peer-to-peer systems have been used in many application domains, the architecture was popularized by the file-sharing system Napster, which was originally released in 1999 (Hess et al. 2002). In modern peer-to-peer systems, participants are often divided into groups, depending on their qualifications, which take on specific tasks. Thus, the core component of all modern peer-to-peer architectures, which are usually already implemented as an overlay network on the Internet, is a second internal overlay network, which usually consists of the most suitable computers in the network and takes over the organization of the other computers and the provision of the search function.

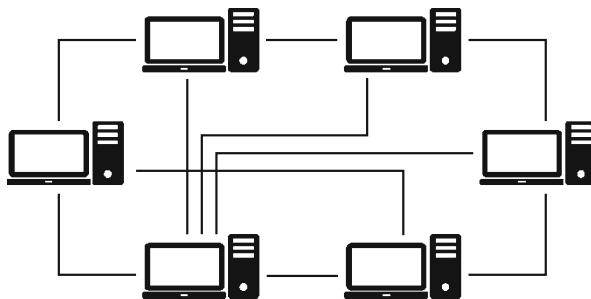


Fig. 2.13 A peer-to-peer network

Generally, two peer-to-peer architecture types can be differentiated (Schollmeier 2001). Unstructured peer-to-peer networks do not impose a particular structure on the overlay network, but are formed by nodes that randomly form connections to one another (Lv et al. 2002). In contrast, in structured peer-to-peer networks, the overlay

is organized into a specific topology, and a protocol ensures that any node can efficiently search the network for a file or resource, even if this resource is extremely rare (Dabek et al. 2003).

Peer-to-peer architectures can be applied for many purposes, the most common of which is content distribution. This includes software publication and distribution (e.g., file-sharing), content delivery networks (e.g., for Web caching or server load balancing), or streaming (e.g., peer casting), which facilitates on-demand content delivery. Other application domains involve science, search, and communication networks.

Peer-to-peer architectures are a key issue in the controversy about network neutrality, a principle that advocates no restrictions on Internet content, formats, technologies, equipment, or modes of communication. Proponents of peer-to-peer architecture argue that governments and large Internet service providers are able to control Internet use and content by directing the network structure toward a client-server architecture. This sets up financial barriers for individuals and small publishers looking to access the Internet and results in inefficiencies for sharing large files.

Assessing some of the advantages and weaknesses of peer-to-peer architectures involve comparisons with client-server architectures. Peer-to-peer architectures have clients with resources such as bandwidth, storage space, and processing power. As more demand is placed on the peer-to-peer system through each node, the system's capacity increases, because new nodes that place demand on the system are also required to share their resources. This accounts for the enormous increase in system security and file verification mechanisms, making most peer-to-peer architectures highly resistant to almost any type of attack. By comparison, a typical client-server architecture shares demands, but not resources. As new clients join the system, the given number of resources must be shared between an increasing number of clients.

2.4.4 Model View Controller Architectures

Another very common architectural pattern, especially in Web-based IS, is the model view controller (MVC). The MVC concept was first described in 1979 for Smalltalk user interfaces by Trygve Reenskaug (Reenskaug 1979), who was working on Smalltalk at Xerox PARC. MVC is often used for the rough design of many complex software systems, sometimes with differentiations and often with several modules divided according to the MVC pattern. As the name suggests, when this architectural pattern is implemented, an IS is divided into three subcomponents: model, view, and controller. The model contains data represented by the presentation and the operations associated with that data. It is completely independent of presentation and control. The presentation is responsible for the representation of the model data and for the realization of user interactions. It knows the model whose data it presents, but is not responsible for processing this data. Further, it is independent of the control system. The presentation is informed about changes to the data and can

then update the representation. The controller manages the presentation and the model. The presentation informs it about user interactions, evaluates these, and then adjusts the presentation and changes the data in the model. In some modern implementations of the MVC pattern, the controller no longer directly updates the data in the model, but indirectly updates the data by accessing the business logic implemented in the model. In a special case of the MVC pattern, the controller can also simultaneously manage multiple presentations or multiple models. Fig. 2.14 provides a conceptional overview on the interactions between the model, view, and controller in an MVC architecture.

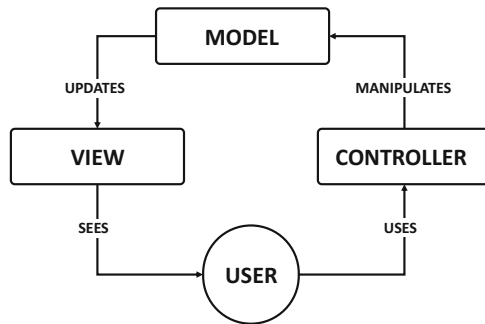


Fig. 2.14 The conceptional interactions in an MVC architecture

The MVC pattern’s main objective is a flexible program design that facilitates later modification or extension and enables the re-usability of the individual components (Krasner and Pope 1988). It is then possible, for instance, to write an application that uses the same model and then makes it available to Windows, Mac, or Linux. While the implementations use the same model, only the controller and the view must be re-implemented.

2.4.5 *Service-Oriented Architecture*

The basic goal of service-oriented architecture (SOA) is to increase the re-usability of business processes by encapsulating these processes or their subprocesses into individual, automated services that can be integrated by more than one client application (Alonso et al. 2004; Papazoglou 2012). SOA describes an abstract IS architecture that delivers services to clients via published and discoverable interfaces. A service in the SOA context can be described as a “representation of a repeatable business activity that has a specified outcome, that is self-contained, that may be composed of other services, and that is a ‘black box’ to consumers of the service.” (The Open Group 2009). Besides a predictable functionality, services are

not restricted concerning their granularity. A service may perform a simple business function or a highly complex process that is composed of multiple other services.

Surrounding this service concept, SOA describes an infrastructure that facilitates the discovery and use of services, while maintaining the loose coupling between service providers and consumers (Huhns and Singh 2005). At a high abstraction level, SOA consists of three main components: (1) the service provider, (2) the service broker, and (3) the service requester. The service provider hosts services and exposes interfaces that allow a service requester to access these services. The description of these service interfaces and all other means necessary to access the service are then published on registries, which are called service brokers. A service requester can now query the broker's repository to find a service offering with matching characteristics. Once a matching service is found, the service broker passes the required information about the service to the requester, who can then bind the service interface and can invoke the underlying service. Based on such an infrastructure, it becomes possible to have multiple providers offering the same service as well as to add or replace different service providers of the same service without affecting the service requester. This dynamic publish-find-bind triangle allows one to construct highly flexible and reliable IT infrastructures, which is ideal for the composition or integration of heterogeneous IS (Georgakopoulos and Papazoglou 2006). Further, it lowers the effort required involved in the provision and consumption of services via the Internet. SOA and the closely related concept of Web services are outlined in some detail in Chapter 6.

Summary

IS are complex systems composed of different, interrelated elements. To handle this complexity, developers utilize representations of IS architectures. Generally, an IS's architecture is defined as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." (ISO/IEC/IEEE 2011). While developing and maintaining an IS, performing an architectural analysis can have various purposes. First, IS architecture can be a tool to design a system such that it is able to meet its non-functional and functional requirements. Second, IS architecture models can also serve as a tool to document the components and their interrelationships of an existing IS. In either case, IS architectural models serve as a shared language that enable all stakeholders to be understood. Thus, they allow stakeholders to reason about a IS's structural properties and facilitate communication about complex dependencies between various physical and virtual components.

Different architectural views can address the differing information needs of stakeholders, which include software developers, who need to understand the software's underlying architecture and interfaces to adjacent systems, or IT and business managers, who require an overview of a firm's application landscape in order to take investment decisions. Kruchten (1995) proposed his famous 4+1 architectural view

model, which has four fundamental architectural views that are interrelated via scenarios and use cases: (1) the logical view (object-oriented decomposition), (2) the process view (process decomposition), (3) the development view (subsystem decomposition), and (4) the physical view (mapping the software to the hardware). Regardless of the IS under consideration, certain principles can be applied to every IS architecture. It is important to keep these principles in mind, because they help one to understand why IS architectural models can look very different although they represent the same system. In this chapter, I proposed nine principles.

Another purpose of explicitly stating an IS architecture is that it may provide best practices and lessons learned that can be re-used on a larger scale or that can be transferred to a different application context. Thus, IS architects aim to capture their knowledge in so-called architectural patterns. These are referred to as abstract descriptions of a recommended architectural approach that has been tested and proven in different IS and environments. Probably the most fundamental and important architectural pattern with regard to designing Internet-based applications is the client-server architecture. The basic idea of client-server is to partition tasks or workloads between the providers of a resource or service (servers) and service requesters (clients). To better understand and conceptualize which functions are performed by which participants in the IS, architects often decompose IS into layers, which represent a logical grouping of functions. The most common classification contains three layers: (1) the presentation layer, (2) the application layer, and (3) the data management layer (Alonso et al. 2004). While a layer is a logical structuring mechanism for the elements that make up a software solution, a tier refers to the physical structuring mechanism for an IS's infrastructure. During the last decades, IS architects have designed various IS that distribute the IS's layers on the tier hardware to different degrees. One-tier, two-tier, three-tier, and multi-tier architectures all have their own characteristics, strengths, weaknesses, and suitable use cases. Another architectural pattern is peer-to-peer. It is a commonly used architectural pattern in which each participant (a participant of the network is often referred to as a peer or a node) has the same capabilities and responsibilities. In Web-based IS, the model view controller architectural pattern is particularly popular. It divides the system into three subcomponents: model, view, and controller. In MVC architectures, every subcomponent has its own tasks and purposes. The model is responsible for managing the application's data, the view handles the presentation of the model in a particular format, and the controller responds to the user input and performs interactions on the data model objects. Finally, SOA is an architectural pattern built around the idea that services should be provided to clients via published and discoverable interfaces. Generally, SOA lowers the effort required involved in the provision and consumption of services via the Internet and thus allows to construct highly flexible and reliable Internet-based applications.

Questions

1. What are the main purposes of IS architecture analysis?
2. What are the nine principles of IS architecture?
3. What are architectural views and why is it important to perform architectural analysis from different perspectives?
4. How are tasks and workload distributed in a client-server architecture?
5. What is the main difference between a client-server architecture and a peer-to-peer architecture?
6. What is the fundamental concept behind service-oriented architecture?

References

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) *Web services: concepts, architectures and applications*. Data-centric systems and applications, 1st edn. Springer, Berlin, pp 123–149
- Avgeriou P, Zdun U (2005) Architectural patterns revisited: a pattern language. Paper presented at the 10th European conference on pattern languages of programs, Isee, 6–10 July 2005
- Boh WF, Yellin D (2006) Using enterprise architecture standards in managing information technology. *J Manag Inf Syst* 23(3):163–207
- Buschmann F, Henney K, Schmidt DC (2007) *On patterns and pattern languages*. Pattern-oriented software architecture, vol 5. Wiley, Chichester
- Castro J, Kolp M, Mylopoulos J (2002) Towards requirements-driven information systems engineering: the Tropos project. *Inf Syst* 27(6):365–389
- Dabek F, Zhao B, Druschel P, Kubiawicz J, Stoica I (2003) Towards a common API for structured peer-to-peer overlays. Paper presented at the international workshop on peer-to-peer systems, Berkeley, CA, 20–21 Feb 2003
- Dropbox (2019) Under the hood: architecture overview. <https://www.dropbox.com/business/trust/security/architecture>. Accessed 29 May 2019
- Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Boston, MA
- Georgakopoulos D, Papazoglou MP (2006) Overview of service-oriented computing. Paper presented at the 4th international conference on service oriented computing, Chicago, IL, 4–7 Dec 2006
- Golden B (2013) *A unified formalism for complex systems architecture*. Ecole Polytechnique, Palaiseau
- Hess T, Anding M, Schreiber M (2002) Napster in der Videobranche? Erste Überlegungen zu Peer-to-Peer-Anwendungen für Videoinhalte. In: Schoder D, Fischbach K, Teichmann R (eds) *Peer-to-peer*. Xpert.press. Springer, Berlin, pp 25–40
- Huhns MN, Singh MP (2005) Service-oriented computing: key concepts and principles. *IEEE Internet Comput* 9(1):75–81
- ISO/IEC (2005) Information technology – open distributed processing – unified modeling language (UML) version 1.4.2. <https://www.iso.org/standard/32620.html>. Accessed 15 Sept 2019
- ISO/IEC/IEEE (2011) Systems and software engineering – architecture description. <https://www.iso.org/standard/50508.html>. Accessed 15 Sept 2019
- Kircher M, Jain P (2013) *Pattern-oriented software architecture, patterns for resource management*. Wiley software patterns series, vol 3. Wiley, Chichester

- Krasner GE, Pope ST (1988) A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J Object-Oriented Program* 1(3):26–49
- Kruchten P (1995) The 4+1 view model of architecture. *IEEE Softw* 12(6):42–50
- Lankhorst M (2017) *Enterprise architecture at work*. The enterprise engineering series, 4th edn. Springer, Berlin
- Lins S, Thiebes S, Schneider S, Sunyaev A (2015) What is really going on at your cloud service provider? Creating trustworthy certifications by continuous auditing. Paper presented at the 48th Hawaii international conference on system sciences, Kauai, Hawaii, 5–8 Jan 2015
- Lins S, Grochol P, Schneider S, Sunyaev A (2016a) Dynamic certification of cloud services: trust, but verify! *IEEE Secur Priv* 14(2):66–71
- Lins S, Schneider S, Sunyaev A (2016b) Trust is good, control is better: creating secure clouds by continuous auditing. *IEEE Trans Cloud Comput* 6(3):890–903
- Lv Q, Cao P, Cohen E, Li K, Shenker S (2002) Search and replication in unstructured peer-to-peer networks. Paper presented at the 16th international conference on supercomputing, New York, NY, 22–26 June 2002
- Papazoglou MP (2012) *Web services and SOA: principles and technology*, 2nd edn. Pearson, Harlow
- Reenskaug T (1979) Thing-model-view-editor: an example from a planning system. <http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf>. Accessed 15 Sept 2019
- Schmidt DC, Stal M, Rohnert H, Buschmann F (2013) *Pattern-oriented software architecture, patterns for concurrent and networked objects*. Wiley software pattern series, vol 2. Wiley, Chichester
- Schollmeier R (2001) A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Paper presented at the 1st international conference on peer-to-peer computing, Linköping, 27–29 Aug 2001
- Sommerville I, Sawyer P (1997) *Requirements engineering: a good practice guide*. Wiley, New York
- Steinmetz R, Wehrle K (2004) Peer-to-peer-networking & computing. *Informatik-Spektrum* 27(1):51–54
- The Open Group (2009) *SOA source book*. Van Haren Publishing, Zaltbommel
- Winter R, Fischer R (2006) Essential layers, artifacts, and dependencies of enterprise architecture. Paper presented at the 10th IEEE international enterprise distributed object computing conference workshops, Hong Kong, 16–20 Oct 2006
- Zachman JA (1987) A framework for information systems architecture. *IBM Syst J* 26(3):276–292

Further Reading

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) *Web services: concepts, architectures and applications*. Data-centric systems and applications, 1st edn. Springer, Berlin, pp 123–149
- Sommerville I, Sawyer P (1997) *Requirements engineering: a good practice guide*. Wiley, New York

Chapter 3

Design of Good Information Systems Architectures



Abstract

Each information system (IS) has an underlying architecture, although its complexity and scope can vary quite substantially for different kinds of systems. Since design decisions about the architecture define the very foundation of an IS, the design decisions cannot be easily undone or altered after they were made. If not taken seriously enough, improper IS architecture designs can result in the development of systems that are incapable of adequately meeting user requirements. Understanding the concept of good IS architecture design and taking design decisions diligently is, therefore, highly important for an IS development project's success. In order to answer the question of what constitutes a good IS architecture, this chapter examines the importance of design decisions across a system's lifecycle. In particular, two different perspectives on the concept of good IS architecture design are explicated: (1) design as the process and (2) design as the outcome of a design process. The two perspectives are closely related to each other and generally help explain the more abstract concept of IS architecture design and particularly the characteristics of a good IS architecture.

Learning Objectives of this Chapter

This chapter's main learning objective is to understand what constitutes a good IS architecture. After studying this chapter, the readers will be able to distinguish between the two main perspectives on the design concept and assess the IS architecture's goodness from these two angles. The readers will particularly gain a general understanding of how the IS architecture's quality can be measured and evaluated. Furthermore, this chapter intends to impart knowledge on how the design process needs to be shaped to create an architecture design that fulfills its intended purpose and can be considered successful.

Structure of this Chapter

This chapter on designing good IS architectures is structured as follows: The first subchapter provides a general introduction to the goodness concept in the IS architecture design context, as well as a general introduction to the two main perspectives, i.e. process and outcome. The second subchapter explains the outcome perspective in detail by clarifying the role of different types of IS architecture

requirements and presenting different quality criteria for assessing the fulfillment of these requirements. The third and final subchapter explains, from the process perspective, which activities are involved in the process of designing a good IS architecture, what methods can be applied to support the process, and how the success of the design project may be determined.

3.1 Architecture Design

As the previous chapter clarifies, a system architecture refers to a set of design decisions about a system. This chapter's focus now shifts toward answering the question of what constitutes a good IS architecture and the importance of design decisions across a system's lifecycle, as well as their impact on the long-term viability and success of the IS. To lay the foundation for this chapter, the following paragraphs provide a basic understanding, to a certain extent, of the terms and concepts related to the IS architecture design and, subsequently, delineate what constitutes *good* design in the context of IS architectures and Internet computing.

The notion of *design* is ambiguous in the sense that the term is used differently across different contexts. This chapter will, therefore, highlight its meaning in the context of IS architectures. The term *design* has a dual meaning: First, *design* refers to the process that defines a system's or component's architecture, which includes the design of its components, as well as the design of its interfaces within and across the system or component (IEEE 1990).

Architecture Design (Process Perspective)

Architecture design refers to the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of an information system (IEEE 1990).

Second, and beyond understanding design as a process of defining the system's or its components' IS architecture, the term *design* inherently involves a second perspective, which is the design as the *outcome* of the design process (Antony et al. 2009).

Architecture Design (Outcome Perspective)

Architecture design refers to the architecture design process's outcome, that is the collection of hardware and software components and their interfaces, which makes up the framework for the development of an information system (IEEE 1990).

The two perspectives are closely related to each other, which is why researchers and practitioners alike often have difficulty distinguishing between them (Majidi et al. 2010). For instance, the system's quality attributes, such as functional suitability, reliability, security, and usability—to name a few—can be considered as the architecture design's characteristics that will be shaped by the design decisions taken over time by developers and system architects. Critically important at this juncture is the notion that IS architecture design is not a one-time effort or phase with a clearly defined set of activities and well-defined start and end points. Instead, designing is a continuous activity that spans across the system's entire lifecycle. The emphasis here is that, although there is indeed often a phase during which the architecture design is more in focus, the design is certainly not limited to this particular phase (Taylor et al. 2009).

At its core, IS development is an engineering discipline that seeks to create useful artifacts (e.g., applications and systems) to support various purposes across industries and contexts. A *good* IS architecture is, thus, an architecture that is *fit for purpose*, which means that it supports the system's intended goals and allows for the implementation of required features and behaviors (Bass et al. 2012). Please note that there is no such thing as a system without architecture (Taylor et al. 2009; Bass et al. 2012). Perhaps the architectural representation is not immediately obvious to the stakeholders involved with the development or use of the system (Bass et al. 2012). Design decisions might be implicit in the choice of programming language, integrated development environment, or context (i.e., Internet computing). Yet, all systems have an underlying architecture, even though the complexity and adequateness can vary quite substantially for different kinds of systems.

This chapter argues that the two perspectives introduced above (i.e., the outcome perspective and the process perspective) are helpful tools to generally explain the more abstract concept of IS architecture design (Bass et al. 2012) and particularly what the characteristics of a *good IS architecture* are. On the one hand, from an outcome perspective, designing an IS architecture is about doing the right things, that is designing a system that seeks to fulfil its intended purpose, which is specified through its functional and quality requirements and subsequently or iteratively translated into features and system behaviors (Bass et al. 2012; Tang et al. 2006). Against the backdrop of a broad variety of software in the context of Internet computing, it becomes apparent that there is no single architecture or design that fits with every purpose. This notion of the IS engineering discipline is also the reason why there is no such thing as a bad architecture per se. Instead, an architecture that does not fit with the intended purpose may be considered bad architecture. Yet, there might be situations in which an architecture design fits particularly well with the overarching goals or requirements, while in other situations, this is not the case (Taylor et al. 2009). For instance, an application that reads sensor data from a nuclear plant to control the cooling system's temperature might be more demanding in terms of stability, security, and performance than a sensor-based application that checks the fill levels of smart bins in a shopping center. Consequently, the underlying architecture has to be designed in different manners to account for these requirements at runtime.

On the other hand, the process perspective describes important aspects that help with the design of good IS architectures. This perspective seeks to focus on the design process itself and highlights important levers that can assist developers and architects with creating good IS architectures. The process of designing an IS architecture considers the *how* and *why* design decisions that were made throughout the lifecycle of a system. From a more practical point of view, this process is about *doing things right*. Chapter 3.3 will, therefore, highlight important principles of the IS architecture design process.

Overall, well-designed IS architectures have a number of benefits that explain why it is important to diligently take design decisions. To start with, design decisions about the architecture constitute the very foundation of a system. These decisions cannot be easily undone or altered once they were made. Moreover, extensive investments of time and effort to create a new architecture or refactor existing architectures are often neither feasible nor reasonable. If not taken seriously enough, improper IS architecture designs can result in the development of systems that are incapable of meeting user requirements in an adequate manner. Hence, deficiencies in the architecture, especially in the early stages of the software development process, can have negative effects on important system behaviors in future. For this reason, early design decisions are critical and characteristically require IS developers and dedicated architects to anticipate future demands and account for these demands in the architecture design to the best of their ability. This is particularly true for more complex and larger applications (Medvidovic and Taylor 2010). In the context of Internet computing, a well-designed architecture is often critical for its ability to meet demands concerning performance efficiency, security, or reliability.

Moreover, a well-designed architecture often leads to higher quality software (Medvidovic and Taylor 2010). Leveraging established, tried-and-tested guidelines, as well as prior knowledge and experience, for architecture design can increase the overall quality of the IS—and the likelihood of its success. Importantly, similar types of systems, such as Internet-based applications with similar requirements and characteristics, can benefit from such knowledge on designing and implementing a solid architectural foundation (Medvidovic and Taylor 2010). In sum, a good IS architecture “facilitates application systems development, promotes achievements of system’s functional requirements, and supports reconfiguration” while “[...] a bad architecture [...] can thwart all three objectives” (Hayes-Roth et al. 1995, p. 288).

Despite these benefits of well-designed IS architectures, there are generally several drawbacks of IS architectures. Challenges commonly noted by practitioners and IT managers include the IS architectures’ focus detracting from the system’s more important aspects, especially those concerning the system’s functionality and practical business use. The argument here is that a well-designed IS architecture is not of much use unless it enables the achievement of business goals. This aspect becomes even more pronounced in an Internet computing age in which technological change is pervasive.

Well-known examples for the importance of architecture designs in an era of Internet computing are digital platform ecosystems, such as Apple’s iOS and

Google's Android. For instance, Apple's ecosystem includes software components (e.g., iOS, third-party applications) and hardware components (e.g., iPhone, iPad). The platform architecture denotes the platform's conceptual blueprint and design rules that describe how the platform's components (e.g., its core technology and adjacent third-party applications) interrelate (Tiwana et al. 2010; Ghazawneh and Henfridsson 2013). This ecosystem's success depends on careful interplay between the software and hardware components, which is ingrained in the ecosystem's architecture design. Arguably, the architecture has important implications for how consumers and third-party developers use the ecosystem. On the one hand, if the architecture is too restrictive, there might be less incentives for third-party developers to create innovative apps, which, over time, can lead to the demise of the system itself. The resulting lack of innovative apps will urge consumers to seek alternative platforms that better address their needs. On the other hand, if the architecture is too open and flexible, the apps' heterogeneity increases complexity and governance efforts, while decreasing the whole ecosystem's maintainability, which can eventually have the same drastic effects on the platform's long-term success.

By and large, the digital platform example emphasizes the importance of understanding why architecture is an important concept in Internet computing and why a good IS architecture creates the very foundation of success and growth. As the evolution of Apple's ecosystem over the past decade also suggests, architecture design decisions are not carved in stone (Tiwana 2015). Instead, such architectures evolve over time, which is why change is an inherent aspect of many of today's system architectures. Hence, the process of developing architectures is not completed after its first definition. This example also highlights that the architecture designs' *goodness* concept depends on the definition of success. In the case of digital platforms, success might be described as the platform's ability to be "generative and evolvable in order to survive in the long run" (de Reuver et al. 2018, p. 130). To achieve this, the architecture's quality attributes, such as maintainability, flexibility, and compatibility might be more important for this type of architecture than for standard IS used in a standard organizational context.

The following two subchapters explain the outcome perspective (see Chapter 3.2) and process perspective (see Chapter 3.3) in more detail.

3.2 IS Architectures' Quality

Starting with the outcome perspective on the design of good IS architectures, the following chapter focuses on explaining the aspects that enable evaluating the IS architectures' aptness for their intended purpose. To this end, the concept of requirements expressing the IS's desired functional and non-functional properties are discussed first. Subsequently, the quality attributes for evaluating the goodness of IS architectures and IS based on these architectures are presented.

3.2.1 *Functional and Nonfunctional Requirements*

As stated, IS architectures are designed with a strong focus on functionality to serve a particular purpose. Beyond this, other requirements are imposed on an IS architecture do not describe a specific function, but the system's properties or behavior instead. Consequently, two type of IS architecture requirements can be distinguished: *functional* and *nonfunctional* requirements.

Functional requirements include all functionalities an IS architecture should provide, for example, a user login or an item search. Fulfilling all functional requirements should eventually enable the IS architecture to be used for a specific purpose. Although functional requirements determine the most obvious parts of an IS architecture, functional requirements do not dictate the specific IS architecture's design. While there are often multiple means to fulfil a requirement, not all of them are equally suitable for the particular use case. Hence, the decision for a particular IS architecture variant is crucial to achieve a certain degree of goodness. For example, the functional requirement to store data in a database could be met with different types of databases (e.g., MySQL, PostgreSQL, MongoDB), which each differs in terms of data organization and the set of features offered. Consequentially, even though the same functional requirements are fulfilled, the different implementation variants have different characteristics, which might be more or less advantageous for the architecture's purpose and often affect the nonfunctional system properties, such as interoperability, maintainability, and security.

Functional Requirement

Functional requirements define the desired features and functions of a system or one of its components. A functional requirement includes the definition of a functionality and its transformation from an input into a desired output (Davis 1993).

Nonfunctional requirements do not relate to *what* functionality a system should provide, but define *how* a system should behave. Consequently, a nonfunctional requirement defines a system property that relates to the system's installation, operation, or maintenance. For instance, the IS may have to handle a highly fluctuating workload, which is why its architecture needs to be either highly scalable or easily migratable to another infrastructure. The system's ability to scale its computational or storage capacity and the ability to be migrated to more suitable infrastructure are not directly the IS architecture's functions, but rather system properties that enable the system's continuous or efficient operation. Due to the fact that functional and nonfunctional requirements strongly influence the usability and viability, a good architecture needs to meet functional and nonfunctional requirements.

Nonfunctional Requirements

Nonfunctional requirements are requirements that are not specifically concerned with the system's functionality, but rather define general quality attributes and constraints (Kotonya and Sommerville 1998).

3.2.2 Quality Attributes

In order to evaluate that the system fulfills the previously defined functional and nonfunctional requirements, these requirements need to be made assessable in an unambiguous, measurable, and testable manner. A requirement is, therefore, usually represented by a single or a combination of several quality attributes that act as a proxy for the requirement (e.g., a system's response time in milliseconds). Whether a requirement is fulfilled or not can then be measured by specifying a value or value range, which needs to be met by the IS architecture's implementation (e.g., the system must respond to requests within 0.3 seconds). Once a quality attribute is associated with a specific value or value range, it is called a quality criterion. With the help of these measurable criteria, the IS architecture's quality can be assessed according to the degree to which the architecture's implementation output meets the value range that aligns with its functional and nonfunctional requirements (IEEE 1992). A well-established framework for software-based systems' quality attributes is the *ISO/IEC system/software product quality model* (ISO/IEC 2011). This model comes with eight categories of characteristics, namely (1) *functional suitability*, (2) *compatibility*, (3) *maintainability*, (4) *performance efficiency*, (5) *reliability*, (6) *security*, (7) *portability*, and (8) *usability* (see Table 3.1). Each of these categories features multiple sub-characteristics, which are defined by means of externally observable features for each outcome. In the following, the categories of the system/software product quality model and selected quality attributes are explicated.

Functional Suitability

Functional suitability is the degree to which an IS architecture offers functions to meet expressed and implied needs for the system's use under specified conditions (ISO/IEC/IEEE 2017a). An IS architecture should provide an infinite number of different functionality combinations, for example, a search function for products in a web shop or a functionality to subscribe to a newsletter. However, the functionality of such ISs can be assessed using the following three quality attributes: (1) functional completeness, (2) functional correctness, and (3) functional appropriateness.

Functional completeness refers to covering all the defined tasks and user objectives a set of functions provides (ISO/IEC 2011). Functional completeness can be evaluated by comparing the functionality implemented in the IS architecture with the planned functionality defined in the specification book. The specification book includes all the functionalities a planned IS architecture must provide and other quality criteria the IS architecture should eventually meet.

Functional correctness extends functional completeness, as functional correctness expresses the degree to which the IS's set of integrated functions produces the correct results with the required degree of precision (ISO/IEC 2011). For example, functional correctness can be illustrated by the development of a calculator. In digital systems, numbers are expressed based on binaries (e.g., $5_{10} = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 101_2$). Due to limitations in the accuracy of digital number representations, computers need to round numbers to a certain degree. Owing to resulting rounding errors, a digital calculator may calculate slightly different results than an analog calculator, which can calculate with more accuracy. If the digital calculator's deviation is not within the tolerance defined in the specification book, the designer of the calculator must revise the calculator's design.

If the requirements on functional completeness and functional correctness are fulfilled, the manner in which the IS influences its users' workflow should be investigated. Therefore, *functional appropriateness* is also relevant and describes the “degree to which the functions facilitate the accomplishment of specified tasks and objectives” (ISO/IEC 2011). Functional appropriateness is related to efficiency when working with the system. For example, systems should hide irrelevant input fields in an input form to make it easier for users to find mandatory input fields.

Compatibility

Compatibility is the degree to which an IS can successfully exchange data with other ISs, while sharing the same hardware or software environment (ISO/IEC 2011). A software environment is not restricted to a single terminal device, but refers to all interacting devices using a particular software. However, compatibility is predominantly about exchanging data, but not about the receiving IS interpreting the data. For example, if geospatial data—including height, width, length—is sent from a server to a smartphone, the data can be received by the smartphone, but it is not necessarily a given that the smartphone software is able to assign the three numerical values to the expression of height, width, or length. A specific structure must, therefore, be implemented and mechanisms are required that enable the geodata's unambiguous interpretation. Interoperability is the attribute that captures this system aspect. *Interoperability* extends the compatibility between two or more systems, such that, after the data exchange, another system can also interpret the data unambiguously (ISO/IEC/IEEE 2017a). Interoperability deals with the structuring of data (syntax) and the transmission of meaning with data (semantics). To convey the data's meaning, the metadata (data about the data) are added during structuring. Metadata are each linked to a data element and follow a defined vocabulary. The use of the vocabulary is defined in a grammar. The metadata are transferred with the data itself and form a self-describing information package that does not depend on any IS architecture. To finally interpret the information package, the receiving system must know the sending system's vocabulary and grammar. The common vocabulary, the grammar, and the related connections to an ontology form the basis of the machine interpretation. Syntactic interoperability is a prerequisite for semantic interoperability and refers to the structuring and transmission mechanisms for data. Separators are used to structure the data and the necessary metadata. Separators do not convey any

meaning to the data, but structure the data such that another system can interpret the data. In XML, for example, the separators are Another widely used format is JavaScript Object Notation (JSON) in which the data to be transmitted and interpreted are assigned as values of variables defined by the user (e.g., *variableName: value*). Individual variables are then listed within curly braces separated by commas. The receiving system can extract the individual data using the separators and recognize their meaning based on the attributes (e.g., variable name in JSON).

Maintainability

Maintainability describes the degree of effectiveness and efficiency with which a system can be modified by intended maintainers (ISO/IEC 2011). Modifications can include an IS's corrections, improvements, or updates to adapt to changes in its environment, requirements, or functional specifications. In the IS context, program code and the associated documentation are often the main targets of modifications during maintenance (ISO/IEC/IEEE 2017b). A high degree of IS maintainability depends on multiple factors, such as structuring the IS architecture into reasonable modules, detailed IS architecture documentation, and dependencies on external services or products.

To achieve efficient maintainability, the IS architecture should be composed of distinct modules. *Modularity* is the degree to which an IS is composed of separated components, such that a change to one component has minimal or no impact on other components (ISO/IEC 2011). All modules of an IS architecture represent distinct units, which encapsulate a certain logical or functional unit of the IS architecture and only communicate with other modules via standardized interfaces. Due to the separation of IS architecture modules, dependencies between modules are reduced, as internal changes in one module do not concern the other modules, as long as the interface remains unchanged. For example, a messenger on a smartphone may have a module for the address book and another module for sending messages. The latter module retrieves data from the former module via an interface. However, the interface is the only connection between these modules, which makes it easy to maintain the address book and the messaging module separately. Therefore, *modularity* also contributes to *modifiability*, which is the degree to which an IS can be changed effectively and efficiently without introducing defects or decreasing the prevalent product quality (ISO/IEC 2011). As modularity reduces dependencies on other IS architecture modules, developers must only consider the functionality of the module to be maintained. Furthermore, modularity facilitates and decreases efforts for testing, because only the modified component requires testing. Hence, by improving modifiability, modularity becomes an important factor for creating a maintainable IS architecture.

Performance Efficiency

In the context of IS architectures, performance predominantly considers the fulfillment of requirements in terms of *time behavior* (or latency), *resource utilization*, and *capacity* (ISO/IEC/IEEE 2017a). Performance efficiency compares performance with the amount of resources that are used to achieve a goal under defined conditions.

A system's time behavior comprises the duration between an event (e.g., user requests, clock events, or errors), processing times, and the system's resulting response to the event. For example, when a student wants to find her marks for the latest test, she makes a request to the university server (*event*). The university server checks the student's authorization to retrieve the marks, queries the marks from the database, and generates a structured representation of the nodes, which should be sent to the student's computer or smartphone. Each computation takes a certain amount of time, which is called *processing time*. Finally, the marks' generated structured representation is returned to the student (*resulting response*). If the previous processing ran reliably and correctly, the processing can be optimized to make the IS and the processes more resource-efficient.

Resource-efficiency is strongly related to *resource utilization*, which is the degree to which the amounts and types of resources an IS uses to perform its functions meet requirements (ISO/IEC/IEEE 2017a). Regarding the previous example's database query, the IS at the student's university may be designed to first read all the marks of all the students and then filter the relevant marks during the generation of the structured representation. This approach comes with a large overhead, because all the marks must be buffered until the filtering process is completed, which consumes time and volatile storage. To decrease resource utilization, the university's IS architects may modify the process and read only the relevant data from the database, such that only the student's marks must be buffered. *Capacity* is the "degree to which the maximum limits of a product or system parameter meet requirements" (ISO/IEC/IEEE 2017a).

All systems have performance requirements, even if these requirements are not explicitly defined. For example, when a user makes a search request for a product in an online shop, a response time of one hour is most probably too long. The user would search for another website and finally buy the desired product elsewhere. The IS must commonly respond to multiple events, which occur within a short time period. These multiple events can include, for example, search requests to the mentioned online shop for a certain product, a sign-up request, or a chat message. Presumably the number of requests per second is not equally distributed during the course of the day. During certain periods the number of requests per second increases or decreases. A good IS architecture can handle such load changes dynamically so that the performance does not deteriorate (e.g., longer response times). Therefore, a system's performance is strongly related to its scalability, which is the IS architecture's capability to handle decreasing or increasing amounts (e.g., of transactions per second). There are two types of scalability (Bondi 2000): *structural scalability* and *load scalability*. Structural scalability is the system's ability to expand in a chosen dimension without major modifications to its architecture, for example, the ease of adding new devices to the IS architecture. Load scalability is the system's ability to remain operational with an increasing or decreasing system load put on the system (Bondi 2000). For example, an online shop's response time should not drastically increase when—suddenly—a large number of customers visit the shop, because of a sales event. To ensure an IS's load scalability, the system can be scaled horizontally or vertically. Horizontal scaling entails adding more devices to

the system in order to distribute the overall system load among multiple devices. Vertical scaling is about replacing a single device's components with more powerful components. For example, a server may be scaled vertically by adding a bigger RAM or faster CPU.

Reliability

In order to make an IS's computations and responses verifiable in terms of correct functioning, the term reliability was introduced. Reliability is defined as the probability that a system will meet its functional and nonfunctional requirements over a given period of time under given operating conditions (Knight et al. 2003) or as "a measure of the continuous delivery of correct service" (Avizienis et al. 2004, p. 13).

An important reliability aspect is a system's availability. In order to deliver its functionalities and thereby fulfill its purpose, an IS must be available to its users. *Availability* is a statistical probability of an IS being operational at an arbitrary point in time. Availability thereby allows determining the IS's average up-time, whereby the opposite is referred to as down-time (i.e., the system is temporarily unusable). Availability depends on two factors: the *mean time between failure* (MTBF) and the *mean time to repair* (MTTR). The MTBF is the period between the IS's start of proper operation and the point in time when it fails. The duration of the IS's non-availability, due to regular patching or maintenance without failure, is not considered in the MTBF. The MTTR considers the failed system's duration of repair. The MTTR's measurement starts with the occurrence of an error and the time at which the system is functioning correctly again. The availability A can be calculated as follows:

$$A = \frac{MTBF}{MTTR + MTBF}$$

Using the formula, the probability that the system provides all functionality as proposed, can be calculated (e.g., 99.999 %). The probability for the IS's downtime F is calculated $F = 1 - A$.

Availability can be increased by adding recovery mechanisms. These mechanisms allow an IS to continue operating despite faults or even component failures. Faults in an IS describe unwanted incidents, for example, entering a wrong datatype into a form. Faults can be avoided by certain mechanisms (e.g., data type validation), handled (e.g., by implementing an appropriate error handling), or tolerated. Faults may, however, cause system failures that inhibit the IS operation and decrease its availability. An IS's ability to recover from such undesired incidents is called resilience. In a resilient system, system faults or even component failures do not necessarily deter the IS's functionality. In an IS architecture where, for example, multiple, synchronized replications of the same service run in parallel on physically separated devices, the requests for the service can be forwarded to an operating device should a certain device crash. Users would not know that another device handles their requests for the service, and the system continues operating. However, a system fault, which cannot be handled by the system itself, can deter the system's

functionality, for example, when the service does not respond to requests. Availability may, therefore, be enhanced by features that help avoid, detect, and repair hardware faults. A highly available and reliable IS does not continue and deliver results that include uncorrected, corrupted data. Instead, ISs usually aim to detect and correct the corruption.

Returning to the online shopping scenario—in addition to availability, another attribute influencing an IS's reliability becomes obvious: *accountability*. Accountability is the “degree to which the actions of an entity can be traced uniquely to the entity” (ISO/IEC 2011). In the online shopping example, the IS needs to provide customer data to the seller (e.g., shipping address) after a purchase had been finalized. Furthermore, the order needs to be traceable for the customer, seller, and administrators to cope with potential claims or issues regarding the purchase, for example, if the customer claims that he/she have not ordered a certain product.

Security

Security is a concept with numerous different domain-specific definitions and architectural approaches (Mouratidis et al. 2009; Thiebes et al. 2016). In the IS context, security predominantly refers to systems' compliance with the three attributes contained in the CIA-triad, namely confidentiality, integrity, and availability (Cherdantseva and Hilton 2013). In this chapter, we follow the security definition proposed by the ISO/IEC, which describes security as the degree to which a result protects data in order to ensure the degree of data access according to the types and levels of the requesters' authorization (ISO/IEC 2011). Authorization presupposes authenticating the users and their authenticity, which allows for the system to recognize the users. *Authenticity* is the “degree to which the identity of a subject or resource can be proved to be the one claimed” (ISO/IEC 2011). When the IS recognizes a user's identity, the user can be assigned to certain privileges or roles, which enables users to access certain resources, such as files, or use certain functionality. For example, users sign up at an online shop and create new user accounts. The system assigns a certain role to the user accounts (e.g., administrator, moderator, customer, seller). After a user, being assigned to the customer role, has signed in, the customer can save her billing address, preferred payment method, and shipping address on the online shopping platform. Users assigned to the seller role have even more functionalities than customers. Sellers are allowed to add and remove products from the shop.

The online shopping scenario also highlights another security attribute: *accountability*. Accountability is the “degree to which the actions of an entity can be traced uniquely to the entity” (ISO/IEC 2011). In the example, the IS needs to provide customer data to the seller (e.g., the shipping address) after a purchase had been finalized. Furthermore, the order needs to be traceable for the customer, seller, and administrators, allowing them to cope with potential claims or issues regarding the purchase, for example, if a customer claims that she had not ordered a certain product.

Furthermore, customers usually do not want anybody to know about their purchases and personal information. This exemplifies the importance of protecting

sensitive data, which were collected by the system, against disclosure to unauthorized users or external parties (ISO 1989). The corresponding attribute is called *confidentiality* of the IS. To assess whether a user has permission to read, delete, or update particular data items, the IS architecture usually includes a user authentication mechanism, as mentioned earlier. Furthermore, confidentiality can be increased by assuring that the IS only grants access to sensitive data with the explicit confirmation of the data owner (e.g., the customer).

While confidentiality controls data disclosure, IS security also implies that data should not be modifiable without permission, which refers to another quality attribute, namely integrity. *Integrity* defines aspects related to maintaining and assuring the data's and services' accuracy and consistency over their entire life-cycle (Boritz 2005). Appropriate integrity measures of an IS may, for instance, ensure that users can only read, but not change or update, certain data on a database.

Usability

Usability is the degree to which certain users or user groups can use an IS “to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use” (ISO 2018). Usability includes the following characteristics (Nielson 2012):

- *Learnability*: How easy is it for users to accomplish basic tasks the first time they encounter the system?
- *Efficiency*: Once users have learned the system, how quickly can they perform tasks?
- *Memorability*: When users return to the system after a period of not using it, how easily can they re-establish proficiency?
- *Errors*: How many errors do users make, how severe are these errors, and how easily can users recover from the errors?
- *Confidence and satisfaction*: How pleasant is it to use the system?

In contrast to the previously described quality attributes, usability relates to the interaction of human beings with the services provided by the IS. The IS should be intuitively controllable for the targeted end-users. Referring to the previously introduced online shop example, users will probably not adopt the online shop if they cannot easily find products and information about products. The following rule of thumb states the importance of IS architecture usability in the e-commerce field: *If users cannot find the product, they cannot buy the product either* (Nielson 2012).

Regarding the *learnability* characteristic, users should find learning to interact with an IS easy. The system should provide functionality for specified users “to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use” (ISO/IEC 2011). Learnability also entails that an IS should provide a high degree of failure prevention (ISO/IEC 2011) to help users learn how to use the system. For example, validators are often used in IS online input forms to prevent users from entering wrong data. These validators predominantly check the type of insertion via syntax.

Consequently, when a user enters her email address, it is checked for a pre-fix before the @ symbol and a subsequent suffix, which is a valid domain.

Certain systems are planned for use by a wide audience and, thus, require a universal design. Universal designs are usable by human beings with the widest possible range of abilities, operating within the widest possible range of situations. In order to formalize the IS's requirements such that its usability fits the audience's abilities, *accessibility* has been introduced as a quality attribute. *Accessibility* refers to the design of an IS that can also be used by people with disabilities (IEEE 1992; Nielson 2012). There are two predominant motivators for designing systems with a high degree of accessibility: (1) *enlargement of usership* and (2) *compliance with legislation, regulations, and standards*. An IS can be directly or indirectly accessible. A system is directly accessible if the user can use the system without assistive technologies, such as screen readers or voice control. The concept focuses on enabling access for people with disabilities or special needs, or enabling access using assistive technology (IEEE 1992; Brewer 2001). To achieve a high degree of accessibility the system architects need to consider the types of disability, the regulations associated with certain disabilities, and the IS's functional availability for assistive technologies. Therefore, the system architects need to consider multiple assistive technologies to make the IS compatible with alternative inputs and outputs (e.g., voice recognition).

Generally, there are several approaches to measure an IS's usability, such as the System Usability Scale (SUS) (Brooke 1996) and the USE questionnaire (Lund 2001). Moreover, usability can be assessed without questionnaires by considering the number of errors users make when performing a task, the time and effort to accomplish a task, the successful operations as a percentage of the total operations, and the amount of time lost in order to recover from failures.

Portability

ISs must sometimes be moved from one environment (e.g., a hardware device) to another for different reasons, such as increasing performance by running its software components on a more capable hardware platform. When an IS is moved from one environment to another, multiple issues can occur, such as the new operating system poorly supporting the integrated frameworks and programming libraries, faulty configuration files, and unfulfilled hardware specifications. To facilitate the migration of an IS, its architecture's *portability*, *adaptability*, and *installability* are important properties. From a physical perspective, portability describes the effectiveness and efficiency with which an entire IS or its individual components can be transferred from one environment to another (ISO/IEC 2011). However, portability by itself is often not sufficient to run an IS in another environment, since the system may require modifications. Hence, if an IS is for whatever functional or non-functional reason highly dependent on its operational environment's characteristics, the *adaptability* of its architecture becomes more relevant. Adaptability is the extent to which an IS "can effectively and efficiently be adapted for different or evolving hardware, software, or other operational or usage environments" (ISO/IEC 2011). Finally, on condition that an IS has been

adapted to the targeted environment, its architecture needs to be installed, which is an activity that involves another quality attribute: installability. *Installability* is the degree of effectiveness and efficiency with which an IS “can be successfully installed and/or uninstalled in a specified environment” (ISO/IEC 2011). To ease the installation process, installation wizards are commonly used, which automate large portions of the installation process.

Table 3.1 Overview of Quality Attributes for IS Architectures

Quality Attribute	Definition
Functional Suitability	Describes the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions (ISO/IEC/IEEE 2017a).
Compatibility	Describes the degree to which a product, system, or component can exchange data with other products, systems, or components, and/or perform its required functions, while sharing the same hardware or software environment (ISO/IEC 2011).
Maintainability	Describes the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers (ISO/IEC 2011).
Performance efficiency	Performance in the IS architecture field considers the fulfillment of requirements in terms of time behavior (or latency), resource utilization, and capacity (ISO/IEC/IEEE 2017a).
Reliability	Reliability describes the probability that a system will produce correct outputs up till a given time t.
Security	Describes the degree to which a product or system protects information and data such that persons, or other products, or systems have the degree of data access that is appropriate to their types and levels of authorization (ISO/IEC 2011).
Usability	Describes the extent to which specified users can use a product to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use (ISO 2018).
Portability	Describes the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software, or other operational or usage environment to another (ISO/IEC 2011).

Although all of the previously presented quality attributes can be important for a good IS architecture, it is often not feasible for the IS architectures to simultaneously fulfill all the quality attributes to the highest degree. For example, to increase an IS architecture's availability, multiple replications of the IS architecture are usually set up and run in parallel. Thus, in case a replication crashes, another replication can still uphold the system's operation. However, the operation of multiple systems also increases the challenge of maintaining consistency between the different replications. Owing to this and similar trade-offs, there is usually no one-size-fits-all IS architecture. A good IS architecture always strikes a balance between the different quality criteria that are relevant for a particular purpose.

3.3 The Information Systems Architecture Design Process

3.3.1 Basic Process Activities

An IS architecture should be designed according to sound and rigorous design processes and be based on logical design rationales (Antony et al. 2009). While a diverse set of architecture design process and life-cycle models exist (Hofmeister et al. 2007), all of the IS architecture design methods have a lot in common, as they deal with the same basic problems: maintaining intellectual control over the design of architectures that (1) require involvement of and negotiation among multiple stakeholders, (2) are often developed by large, distributed teams over extended periods of time, (3) must address multiple – possibly conflicting –goals and concerns, and (4) must be maintained for a long period of time (Hofmeister et al. 2007). The architecture design process's primary output is an architectural description. A number of basic activities are usually involved in creating an architecture, using that architecture to realize a complete design and then implementing or managing the evolution of a target system or application (Bass et al. 2012). The six most important activities are summarized in Table 3.2.

Table 3.2 Basic Activities of the IS Architecture Design Process, adapted from Bass et al. (2012)

Activity	Description
1	Making a business case for the system.
2	Understanding the architecturally significant requirements.
3	Designing or selecting the architecture.
4	Documenting and communicating the architecture.
5	Evaluating the architecture.
6	Ensuring that the implementation conforms to the architecture.

All these basic activities rely, to a certain extent, on tapping stakeholders’ knowledge, preferences, and visions about the functions and characteristics of an architecture to be designed (Bass et al. 2012). Consequently, each of these activities includes a similar process of identifying the stakeholders who are relevant for a certain activity, meeting with all of them simultaneously in a room, briefing them about the activity, gathering their needs and opinions, and finally solving the puzzle concerning how to satisfy stakeholders’ needs, given the characteristics and limitations of specific technologies.

Owing to the design task's complexity, these above-mentioned activities might not be executed sequentially. Depending on the overall process model, they may be performed repeatedly, at multiple levels of granularity, in no predictable sequence, and until the architecture is complete and validated (Hofmeister et al. 2007). Generally, one differentiates between three activity management strategies that are known from the software development domain, namely (1) waterfall, (2) iterative and (3) agile processes (Bass et al. 2012).

The waterfall model organizes the design process into a series of connected sequential activities (Isaia and Issa 2015). The process flows in a step-by-step sequence in largely one direction, downwards like a waterfall, through different phases, each with entry and exit conditions and a formalized relationship with its upstream and downstream neighbor phases. By contrast, iterative processes regard the design process as a series of short cycles called iterations (Boehm 1988). Each iteration begins with identifying the current stage's objectives and requirements, as well as analyzing the alternatives and constraints. At the end of each iteration, a design is implemented and tested while the next iteration focuses on the remaining requirements in the sense of iterating toward the given problem's ultimate design. Agile methods focus on the early and frequent delivery of working ISs, close collaboration between developers and customers to ensure satisfaction, and a focus on adapting to changing circumstances, such as evolutionary or late-arriving requirements (Beck et al. 2001). These methodologies usually favor simplicity, i.e. they eschew substantial up-front work, on the assumption that requirements always change and they continue changing throughout the design project's life cycle (Glowalla and Sunyaev 2015). In the following, the six basic process activities introduced above are outlined in more detail.

Activity 1: Making a Business Case for the System

Many IS projects are considered failures, because they did not actually solve a valid business problem or have a recognizable return on investment (Albin 2003). Architects who do not receive clear instructions concerning the problem that must be solved, may end up focusing on solving technical issues that ultimately do not address the original problem. Thus, identifying and specifying a business case should be the starting point for each architecture design process. A business case justifies an organizational investment, describing the overall problem that should be solved and the resulting benefits for an organization and its respective stakeholders (Bass et al. 2012). After project initiation, the business case is constantly reviewed to assess the initial estimates' accuracy and – if necessary – updated to examine new or alternative angles on the opportunity. By documenting the expected costs, benefits, and risks, the business case serves as a basis for the whole design process to ensure that the architecture fulfills the business goals. However, while a business case can neither be determined nor decided solely by an architect, architects need to be involved in the creation of the business case to ensure that they gain the required domain and problem knowledge to create a satisfactory IS architecture design in the subsequent process steps.

Various techniques and tools have been developed to help define business cases. For example, the SWOT analysis can be applied in early project phases to identify strengths, weaknesses, opportunities, and threats related to project planning (Mesly 2016; Chermack and Kasshanna 2007). The SWOT analysis is normally used as a strategic planning technique and considers the strength and weakness within an organization, as well as the external opportunities and threats stemming from community or societal forces (see Fig. 3.1). Transferred to the context of IS architecture design, the architects and various stakeholders should discuss the

project's strengths that give it an advantage over others, while keeping in mind the weaknesses, for example, concerning the available human resources, time, budget, technology limitations, and existing legacy systems. Strengths and weaknesses should be mapped to opportunities for and threats to the organization, such as future trends in IS, legislation, and customer demands in order to determine the business goals that can be achieved by using internal strengths to harness opportunities, and to set up boundaries for internal weaknesses and external threats.

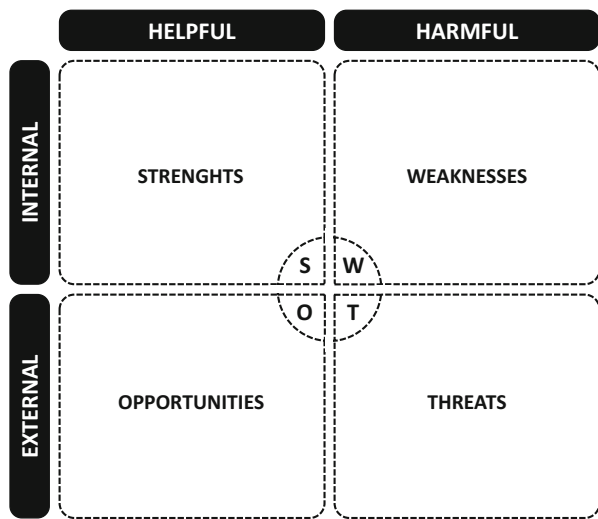


Fig. 3.1 SWOT matrix

Activity 2: Understanding the Architecturally Significant Requirements

After specifying the business case, the architect can start the requirements analysis in close cooperation with the relevant stakeholders of the system to be designed. The requirements analysis is the activity of gathering, identifying, and formalizing the requirements in order to understand the actual problems for which an architecture, as a solution, is sought and to learn the purpose and scope of the future system (Albin 2003). While there are diverse requirements for an IS, an architect should filter the available requirements and only focus on the architecturally significant requirements (ASR), which are the requirements of an IS that influence its architecture (Hofmeister et al. 2007). Thus, this activity consists of analyzing the problem and the stakeholders' needs, extracting the most critical needs from the overarching business problem description, and determining the desired quality attributes and functions of the system that must be built, which, together with the business goals, drive the architecture design decisions (Falessi et al. 2011). As explicated in Chapter 3.2, the requirements analysis usually results in documents comprising a functional specification and nonfunctional specification. Furthermore, the analysis

may already result in a high-level user interface design that depicts how customers, systems, or other stakeholders interact with the system.

Requirements Analysis

The requirements analysis is the activity of gathering, identifying, and formalizing requirements in order to understand the actual problems for which an architecture, as a solution, is sought, and to learn the purpose and scope of the future system (Albin 2003).

There are a variety of techniques for eliciting requirements from the stakeholders, such as use cases, scenarios, finite state machine models, and formal specification languages. The Quality Attribute Workshop (Barbacci et al. 2003) is a famous approach used for analyzing and eliciting the architecturally significant requirements. In the Quality Attribute Workshops, an external or internal team meets with stakeholders and during these meetings they generate, prioritize, and refine scenarios, which represent the quality attribute requirements. From these discussions, architects gain insight into the stakeholder assumptions that may not have been expressed during the elicitation of business goals, and they also gain insight into which quality attributes are pulling the architecture in different directions, thereby informing subsequent architectural tradeoff decisions (Carnegie Mellon University 2018).

Activity 3: Designing or Selecting the Architecture

The designing activity focuses on transforming the requirements specifications into a technically feasible solution (Albin 2003). Architects undergo diverse decision-making activities to fulfill the stakeholders' needs as defined in the previous activity by choosing the most appropriate architecture design option(s) from the available alternatives and using design principles and patterns (Falessi et al. 2011). During the design activity, architects typically have to decide between different candidate architectural solutions. Candidate architectural solutions may present alternative solutions and/or may be partial solutions, i.e. fragments of an architecture, that satisfy an ASR or a set of ASR (Bass et al. 2012). Architects need a reliable and rigorous process for selecting candidate architectural solutions and ensuring that the decisions mitigate risks and maximize profit (Falessi et al. 2011). Poor decision-making at this process stage may entail several difficulties, which, in turn, provoke the selection of a worse alternative. For example, architects and stakeholders may have different interests or concerns and, therefore, different views of the system. As a result, they adopt different vocabularies. They might use a single word to define a quality attribute, such as performance, which means something specific to them (e.g., worst-case latency), but something different to another stakeholder, for example, an administrator might interpret performance as the efficient use of system resources, while an end user might understand performance as the time it takes to learn the system. These differences in interpretation tend to cause misunderstandings (Gilb 2005; Moore et al. 2003), provoking the selection of an unsuitable architecture

design, which, in turn, causes stakeholder dissatisfaction and, eventually, a major rework to satisfy the real stakeholders' needs.

Architects can rely on diverse decision-making techniques, such as the quality attributes importance description (Falessi et al. 2011). Not all the quality attributes are equally important for a stakeholder; certain quality attributes are more desirable than others and, therefore, architects should describe the extent to which a quality attribute is needed.

The design activities' result is a specification of architecture components, such as software objects and their relationships, a specification of how to build the application or system, and a specification of the technical constraints on the implementation. Design solutions are typically specified by using notations, such as those in the Unified Modeling Language (UML). A design specification's level of detail varies. IS architects and software developers generally believe that the amount of effort expended on design is never sufficient (Albin 2003). The architecture design documents should also include information about the design rationale, i.e. commentary on why decisions were made, what decisions were considered and rejected, and the traceability of decisions to requirements (Bass et al. 2012).

Activity 4: Documenting and Communicating the Architecture

There is a growing awareness in industry and academia that effectively documenting and sharing architectural knowledge inside the developing organization and with external actors are key factors for IS project success (Avgeriou et al. 2007; Lago and Avgeriou 2006; Avgeriou et al. 2008). Developers must understand the work assignments that the architecture requires of them, testers must understand the task structure that the architecture imposes on them, management must understand the architecture's scheduling implications, and so forth (Bass et al. 2012). Thereby, the architecture's documentation should be informative, unambiguous, and understandable to many people with varied backgrounds.

Architectural knowledge that should be documented is defined as the knowledge about a software architecture and its environment (Kruchten et al. 2006). The most important type of architectural knowledge is the architectural and architecture design decisions, which shape a software architecture (Jansen and Bosch 2005). Other types of architectural knowledge include concepts from architecture design (e.g., components, connectors) (Tang et al. 2007), requirements engineering (e.g., risks, requirements), people (e.g., stakeholders and their roles), and the development process (e.g., activities) (de Boer et al. 2007).

Architectural Knowledge

Architectural knowledge is defined as the knowledge about a software architecture and its environment (Kruchten et al. 2006), such as architecture design and architectural decisions, which shape a software architecture and concepts from architecture design (e.g., components, connectors).

Documenting architectural knowledge provides several benefits (Jansen et al. 2009), including (1) asynchronous communication, not face-to-face, among stakeholders to negotiate and reason about the architecture, (2) reducing the effect of knowledge vaporization (Jansen et al. 2008), (3) steering and constraining the implementation, (4) shaping the organizational structure, (5) reusing architectural knowledge across organizations and projects, and (6) supporting the training of new project members.

As the architecture must be communicated clearly and unambiguously to all of the stakeholders, a fundamental principle of architecture documentation is “Write for the reader,” meaning that architects need to understand who will read the documentation and how they will use it (Bass et al. 2012). The perfect architecture is useless if it has not been stated comprehensibly. If an architect takes the trouble of creating a good architecture, the architect *must* also go the extra mile of describing it in sufficient detail, without ambiguity, and organized such that others can quickly find the information they need (Clements et al. 2002).

However, when ISs grow in size and complexity, so does the architectural documentation (Jansen et al. 2009). In such large and complex ISs, the documentation often consists of multiple documents, each of considerable size (i.e., up to hundreds of pages). Moreover, it becomes more complex, as within and between these documents, there are many concepts and relationships, multiple views, different levels of abstraction, and numerous consistency issues. Therefore, architects and stakeholders need to spend significant effort in documenting the architectural knowledge and they must be convinced that they will achieve a good return on their investment.

Various techniques and tools have been developed to support the documentation and communication of architectural knowledge, such as the tool suite known as Knowledge Architect that supports architects in creating, using, and managing architectural knowledge across documentation, source codes, and other representations of architectures (Jansen et al. 2009) or indexing techniques that rely on lightweight ontology to improve the retrieval and traceability of knowledge (Tang et al. 2011).

Activity 5: Evaluating the Architecture

An important step involves evaluating the architecture design to determine if it satisfies the ASR (Albin 2003). If the measured or observed quality attributes do not meet their requirements, a new design must be created. This, however, requires that architects must clearly articulate the quality attribute requirements that are affected by the architecture. Thus, having an explicit and well communicated architecture is the first step toward ensuring architectural conformance.

Architecture designs can be evaluated differently. Scenario-based techniques provide one of the most general and effective architecture evaluation approaches. Through using these techniques, an architecture is validated by analyzing how predefined scenarios impact on architectural components (Tekinerdogan 2004). The most mature methodological approach is found in the architecture tradeoff

analysis method (Bass et al. 2012), while the economic implications of architectural decisions can also be explored (Bass et al. 2012).

Similar to software development, there are levels of evaluation, which range from evaluating small, individual pieces in isolation to evaluating an entire system (Bass et al. 2012). Unit testing refers to evaluations focusing on specific pieces of the architecture, such as single components or layers. Ensuring that a unit has passed its unit tests is a precondition for this unit's delivery to the integration activities. Unit tests will not usually identify errors dealing with the interaction between elements and further integration evaluations are, therefore, required. Integration evaluation concentrates on finding problems related to the interfaces between the architectural elements in a design. At the integration evaluation's completion, the architect has confidence that the architecture's elements work together correctly and provide – at least to a certain extent – correct architecture-wide functionality.

Activity 6: Ensuring that the Implementation Conforms to the Architecture

Finally, when an architecture is created and used, it enters a maintenance phase (Bass et al. 2012). Architects cooperating with developers and administrators have to ensure that the actual architecture and its implementation fit together throughout the architecture's life cycle.

When a change affects the architecture, the original architecture design must be updated to ensure that the system remains flexible and continues functioning as originally designed. When an architectural change causes the interactions to become more complex, which, in turn, obstructs changes to the system, the architecture is degenerating. Architectural degeneration is a mismatch between the system's actual functions and its original design (Williams and Carver 2010). Since architectural degeneration is confusing for developers, the system must undergo either a major reengineering effort or face early retirement (Hochstein and Lindval 2005).

Architectural Degeneration

Architectural degeneration is a mismatch between the actual functions of the system and its original design (Williams and Carver 2010).

To address these problems, architects and developers need to better understand the interplay between a change and the architecture prior to making the change (Buckley et al. 2005). An architect should consider the following steps (Williams and Carver 2010):

- *Change understanding and architecture analysis:* A change analysis tool can be used, thereby enabling the developer to analyze a change before implementation in order to understand the change, the architecture, and how the change fits with the architecture (O'Reilly et al. 2003).
- *Build historical baseline of software change data:* In order to gain deeper insight into future changes, architects and developers can record information about the

change (i.e., change impact, difficulty, and required effort) and its impact on the architecture (Lehmann et al. 1998).

- Group changes based on impact/difficulty: Change requests can be grouped based on their characteristics (Williams and Carver 2010). Similar changes should exhibit a similar impact on the system. Heuristics can be developed to handle certain types of changes (Kim et al. 2008; O'Neal and Carver 2001).
- Facilitate discussion among developers: Architects can rely on methods that facilitate discussion among the development team to achieve consensus with regard to the implementation approach. A characterization scheme should facilitate consensus building by providing a list of items for discussion to prevent the change from violating the planned architectural structure (Buckley et al. 2005; Hinley 1996).
- Facilitate change difficulty/complexity estimation: The characterization scheme should allow a developer to determine change complexity as a function of type and size. Characterizing the change request's context, i.e. a description of influencing factors external to the request, should also help facilitate difficulty estimation, because certain types of changes may be more difficult in certain domains than others (Banker et al. 1993; Lam and Shankararaman 1998)

3.3.2 Example Method for Designing Architectures: Attribute-Driven Design (ADD) Method

Researchers and practitioners have developed manifold methods to support the architecture design process. While discussing each method or set of methods is beyond the scope of this chapter, in the following the Attribute-Driven Design (ADD) method is briefly summarized as an example and well-known architecture design method. For a more in-depth description of the ADD method's individual steps and practical guidelines for carrying out each step, please refer to Wojcik et al. (2006).

In general, the ADD method (Wojcik et al. 2006) is an approach to define software architectures by basing the design process on the architecture's quality attribute requirements. The ADD method was developed at the Software Engineering Institute, a U.S. research and development center. ADD follows a recursive design process that decomposes a system or system element by applying architectural tactics and patterns that satisfy a set of ASR (see Fig. 3.2). ADD essentially follows an iterative Plan, Do, and Check cycle: (1) *Plan*: Quality attributes and design constraints are considered in order to select which types of elements will be used in the architecture. (2) *Do*: Elements are instantiated to satisfy quality attribute requirements, as well as functional requirements. (3) *Check*: The resulting design is analyzed to determine if the requirements are met.

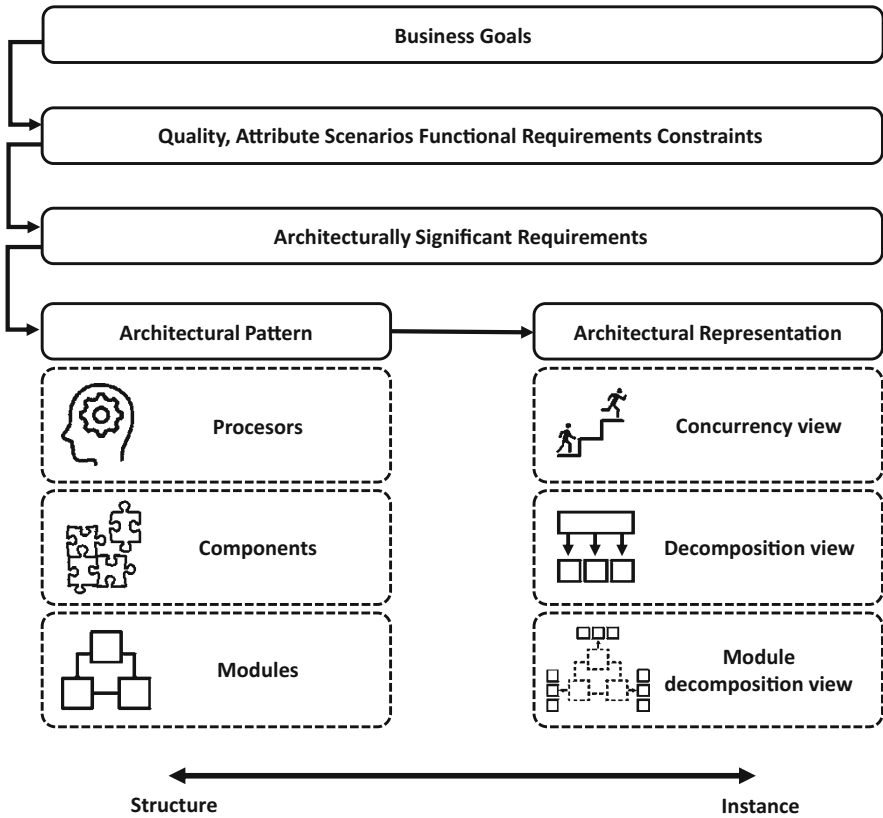


Fig. 3.2 Recursively designing the architecture using ADD (adapted from Hofmeister et al. (2007))

Architects use the following iterative steps when they design an architecture by using the ADD method:

- I. Confirm that there is sufficient information about the ASR to proceed with the ADD. Architects have to ensure that the stakeholders have stated and prioritized the requirements according to business goals.
- II. Choose the module to decompose. The module to start with is usually the whole system. For designs that are already partially completed, i.e. previous iterations via the ADD, the module is an element that is not yet designed. All required inputs for this module should be available (constraints, functional requirements, quality requirements).
- III. Identify the ASR for the chosen module from the set of concrete quality scenarios and functional requirements. This step determines what is important for this module.
- IV. Choose a design concept for the module that satisfies the ASR according to these steps:

- IV.1. Identify the design concerns that are associated with the ASR for the module. For example, for a quality attribute requirement regarding availability, the major design concerns might be fault prevention, fault detection, and fault recovery.
- IV.2. For each design concern, create a list of alternative patterns that address the concern.
- IV.3. Choose an architectural pattern that satisfies or best satisfies the ASR. Create or select the design pattern based on the tactics that can be used to achieve the ASR. Identify child modules that are required to implement the tactics. For example, in a Ping-Echo pattern, the architect has to define child modules that send a ping signal and other child modules that are to respond with an echo signal. Document the rationales for choosing the pattern.
- IV.4. Instantiate child modules and allocate functionality, as well as responsibilities from use cases, and represent the results using multiple views.
- IV.5. Define the child modules' interfaces. Interfaces describe the PROVIDES DATA and REQUIRES DATA assumptions that software elements make about each other. Document this information in each module's interface document.
- IV.6. Verify that the modules and child modules, thus far, meet functional requirements, quality attribute requirements, and design constraints. This step verifies that no important requirement was forgotten and prepares the child modules for further decomposition and implementation.
- V. Repeat the above-mentioned steps for every module that requires further decomposition.

The ADD's output is not an architecture that is complete in every detail, but an architecture in which the main design approaches have been selected and vetted. The ADD's output produces a workable architecture early and quickly, an architecture that can be given to other project teams such that they can begin their work while the architect or architecture team continues to elaborate and refine.

3.3.3 Success of Architecture Design Processes: The Iron Triangle

Designing the IS architectures is an early activity in the development cycle and often difficult to verify before system implementation (Antony et al. 2009). Consequently, correcting any mistakes or omissions in an architecture's design can be costly, and ensuring the success of architecture design projects becomes very important.

While every person has an understanding about what project success and project failure mean, success remains an ambiguous, inclusive, and multidimensional concept whose definition is bound to a specific context (Ika 2009). Project success has

long been considered the ability to stay within time, cost, and quality constraints, becoming known as the time/cost/quality triangle or the Iron Triangle (Fig. 3.3). However, projects have often enough been delivered within time, cost, and quality, only to be considered failures. For example, the Ford Taurus car was completed on time in 1995, but turned out to be a disappointing business experience (Shenhar et al. 2007). At the same time, other projects have exceeded time or cost constraints, but are generally considered successful (Pinto and Slevin 1988), such as the *Elbphilharmonie* in Hamburg, Germany. Stakeholders have distinct vested interests in a particular project and, therefore, the perception of success may also vary across various stakeholders (Bryde and Brown 2005).

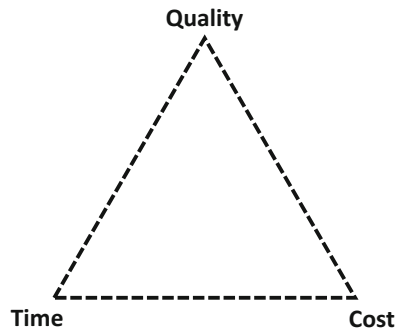


Fig. 3.3 Project management triangle, according to which the optimization of one requirement always comes at the cost of the others

Today, project success is measured by considering diverse dimensions in isolation or combination, such as project safety, customer satisfaction, benefits for organizations, and whether the project meets specifications, is free from defects, conforms to stakeholders' expectations, or minimized construction aggravation, disputes, and conflicts (Toora and Ogunlana 2010; Ika 2009). Owing to trade-offs between the project success dimensions, such as time, cost and quality, a project usually cannot achieve all of them. For example, an architecture might not be of high quality if it is produced within a short period of time and at minimum cost. Thus, before starting the design process, the dimensions have to be specified and prioritized. Throughout the design process, these dimensions should be brought in balance, because they cannot all be achieved optimally at the same time.

Summary

Every IS has an underlying architecture, although its complexity and scope can vary quite substantially for differ kinds of systems. Understanding the concept of good IS architecture design is, therefore, not a simple matter. In order to answer the question

of what constitutes a good IS architecture, this chapter scrutinized the importance of design decisions across a system's lifecycle. In particular, two different perspectives on the concept of IS architecture design were explicated: (1) design as the process that defines a system's architecture, including the design of its components and its interfaces within and across the system and (2) design as the outcome of a design process. The two perspectives are closely related to each other and generally help explain the more abstract concept of IS architecture design and particularly what makes a good IS architecture.

From an outcome perspective, the IS architecture design describes a system that seeks to fulfil its intended purpose, which is specified through its stakeholders' requirements that are subsequently or iteratively translated into features and system behaviors. These requirements can be either functional or nonfunctional. Functional requirements include all the IS architecture's functionalities that enable the architecture to be used for its intended purpose. Nonfunctional requirements, on the other hand, define criteria that can be used to assess a system's operation and maintenance. During the development of the IS architectures, the functional and nonfunctional requirements must be considered to achieve a high quality IS architecture. To evaluate whether the system fulfills the previously defined requirements, these requirements need to be assigned to quality attributes. A measurable quality attribute and for which an acceptable value range is defined, is called a quality criterion. Drawing from the quality criteria, the IS architecture's goodness represents the degree to which the architecture possesses a desired combination of quality criteria. The software quality model provides a structured overview of quality attributes for software-related aspects of an IS architecture. As presented in this chapter, this model comes with six categories of characteristics, namely functional suitability, compatibility, maintainability, performance efficiency, reliability, security, usability.

From the process perspective, the architecture design describes the design process itself and highlights important levers that can assist developers and architects with creating good IS architectures. The process of designing an IS architecture is essentially all about making design decisions throughout the system's lifecycle. Therefore, the IS architectures should be designed via sound and rigorous design processes and based on logical design rationales. These rationales are scattered across different activities involved in creating an architecture. Six of these activities were presented in this chapter: (1) making a business case for the system, (2) understanding the architecturally significant requirements, (3) designing or selecting the architecture, (4) documenting and communicating the architecture, (5) evaluating the architecture, and (6) ensuring that the implementation conforms to the architecture. In addition to these high-level activities, researchers and practitioners have developed different methods that further support the architecture design process. One of these methods presented in this chapter is the Attribute-Driven Design (ADD) method that follows an iterative cycle comprising the specification of quality attributes and design constraints, instantiation of architectural elements to satisfy the attributes and constraints, and evaluation of outcome to determine if the requirements are met.

Closing this chapter, the characteristics of a successful architecture design process were briefly discussed. A traditional method for determining a design project's success is assessing whether the parties involved stayed within the constraints regarding time, cost, and quality, i.e. the Iron Triangle. A more modern approach considers project success by measuring a large variety of dimensions in isolation or combination, such as project safety, customer satisfaction, and whether the project meets specifications. These dimensions have to be specified and prioritized in the early phases of a design project and should be constantly evaluated and brought in balance throughout its duration to maximize project success.

Questions

1. What are the two different perspectives on architecture design?
2. What characterizes a good IS architecture?
3. What types of architectural requirements can be differentiated?
4. Describe three quality attributes of a good IS architecture.
5. What is the role of a business case during the IS architecture design process?
6. Which individuals or groups need to be involved in the IS architecture design process and for what reasons?
7. How can the success of an architecture design process be determined?

References

- Albin S (2003) *The art of software architecture: design methods and techniques*. Wiley, New York, NY
- Antony T, Han J, Vasa R (2009) Software architecture design reasoning: a case for improved methodology support. *IEEE Softw* 26(2):43–49
- Avgeriou P, Kruchten P, Lago P, Grisham P, Perry D (2007) Architectural knowledge and rationale: issues, trends, challenges. *ACM SIGSOFT Softw Eng Notes* 32(4):41–46
- Avgeriou P, Lago P, Kruchten P (2008) 3rd international workshop on sharing and reusing architectural knowledge (SHARK 2008). In: 30th international conference on software engineering, Leipzig, 10–18 May 2008, pp 1065–1066
- Avizienis A, Laprie J-C, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Depend Secure Comput* 1(1):11–33
- Banker RD, Datar SM, Kemerer CF, Zweig D (1993) Software complexity and maintenance costs. *Commun ACM* 36(11):81–94
- Barbacci MR, Ellison RJ, Lattanze AJ, Stafford JA, Weinstock CB, Wood WG (2003) *Quality attribute workshops (QAWs)*, 3rd edn. <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=6687>. Accessed 16 Sept 2019
- Bass L, Clements P, Kazman R (2012) *Software architecture in practice*. Addison-Wesley, London
- Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin R, Mellor S, Schwaber K, Sutherland J, Thomas D (2001) *Manifesto for Agile software development*. <http://agilemanifesto.org>. Accessed 5 May 2019
- Boehm BW (1988) A spiral model of software development and enhancement. *Computer* 21(5):61–72

- Bondi AB (2000) Characteristics of scalability and their impact on performance. Paper presented at the 2nd international workshop on software and performance, Ottawa, ON, 17–20 Sept 2000
- Boritz JE (2005) IS practitioners' views on core concepts of information integrity. *Int J Account Inf Syst* 6(4):260–279
- Brewer J (2001) Access to the world wide web: technical and policy aspects. In: Preiser W, Ostroff E (eds) *Universal design handbook*, 1st edn. MacGraw-Hill, New York, NY, p 66.61
- Brooke J (1996) SUS – a quick and dirty usability scale. In: Jordan PW, Thomas B, McClelland IL, Weerdmeester B (eds) *Usability evaluation in industry*. Taylor & Francis, London, pp 189–194
- Bryde DJ, Brown D (2005) The influence of a project performance measurement system on the success of a contract for maintaining motorways and trunk roads. *Proj Manag J* 35(4):57–65
- Buckley J, Mens T, Zenger M, Rashid A, Kniesel G (2005) Towards a taxonomy of software change. *J Softw Maint Evol Res Pract* 17(5):309–332
- Carnegie Mellon University (2018) The SEI quality attribute workshop. https://resources.sei.cmu.edu/asset_files/FactSheet/2018_010_001_513488.pdf. Accessed 5 May 2019
- Cherdantseva Y, Hilton J (2013) A reference model of information assurance & security. Paper presented at the international conference on availability, reliability and security, Regensburg, 2–6 Sept 2013
- Chermack TJ, Kasshanna BK (2007) The use and misuse of SWOT analysis and implications for HRD professionals. *Hum Resour Dev Int* 10(4):383–399
- Clements P, Garlan D, Bass L, Stafford J, Nord R, Ivers J, Little R (2002) *Documenting software architectures: views and beyond*. Addison Wesley, Boston, MA
- Davis AM (1993) *Software requirements: objects, functions, and states*. PTR Prentice Hall, Upper Saddle River, NJ
- de Boer RC, Farenhorst R, Lago P, van Vliet H, Jansen AGJ (2007) Architectural knowledge: getting to the core. Paper presented at the 3rd international conference on the quality of software architectures, Medford, MA, 11–13 July 2007
- de Reuver M, Sørensen C, Basole RC (2018) The digital platform: a research agenda. *J Inf Technol* 33(2):124–135
- Falessi D, Cantone G, Kazman R, Kruchten P (2011) Decision-making techniques for software architecture design: a comparative survey. *ACM Comput Surv (CSUR)* 43(4):33
- Ghazawneh A, Henfridsson O (2013) Balancing platform control and external contribution in third-party development: the boundary resources model. *Inf Syst J* 23(2):173–192
- Gilb T (2005) *Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using planguage*. Elsevier, Butterworth-Heinemann, Oxford
- Glowalla P, Sunyaev A (2015) Influential factors on IS project quality: a total quality management perspective. Paper presented at the 36th international conference on information systems (ICIS), Fort Worth, TX, 13–16 Dec 2015
- Hayes-Roth B, Pfleger K, Lalanda P, Morignot P, Balabanovic M (1995) A domain-specific software architecture for adaptive intelligent systems. *IEEE Trans Softw Eng* 21(4):288–301
- Hinley DS (1996) Software evolution management: a process-oriented perspective. *Inf Softw Technol* 38(11):723–730
- Hochstein L, Lindval M (2005) Combating architectural degeneration: a survey. *Inf Softw Technol* 47(10):643–665
- Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, America P (2007) A general model of software architecture design derived from five industrial approaches. *J Syst Softw* 80(1):106–126
- IEEE (1990) Standard glossary of software engineering terminology. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=159342&tag=1>. Accessed 16 Sept 2019
- IEEE (1992) Standard for a software quality metrics methodology. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=237006>. Accessed 16 Sept 2019
- Ika LA (2009) Project success as a topic in project management journals. *Proj Manag J* 40(4):6–19
- Isaias P, Issa T (2015) Information system development life cycle models. In: Isaias P, Issa T (eds) *High level models and methodologies for information systems*. Springer, New York, NY, pp 21–40

- ISO (1989) Information processing systems – open systems interconnection – basic reference model – Part 2: Security architecture. <https://www.iso.org/standard/14256.html>. Accessed 16 Sept 2019
- ISO (2018) Ergonomics of human-system interaction – Part 11: Usability: definitions and concepts. <https://www.iso.org/standard/63500.html>. Accessed 19 Sept 2019
- ISO/IEC (2011) Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models. <https://pdfs.semanticscholar.org/57a5/b99e2eff9da205e244337c9f4678b5b23d25.pdf>. Accessed 19 Sept 2019
- ISO/IEC/IEEE (2017a) International standard – systems and software engineering – vocabulary. <https://standards.ieee.org/standard/24765-2017.html>. Accessed 19 Sept 2019
- ISO/IEC/IEEE (2017b) Systems and software engineering – software life cycle processes. <https://www.iso.org/standard/63712.html>. Accessed 19 Sept 2019
- Jansen AGJ, Bosch J (2005) Software architecture as a set of architectural design decisions. Paper presented at the 5th IEEE/IFIP working conference on software architecture (WICSA), Pittsburgh, PA, 6–10 Nov 2005
- Jansen AGJ, Bosch J, Avergiou P (2008) Documenting after the fact: recovering architectural design decisions. *J Syst Softw* 81(4):536–557
- Jansen A, Avgeriou P, van der Ven JS (2009) Enriching software architecture documentation. *J Syst Softw* 82(8):1232–1248
- Kim S, Whitehead EJ, Zhang Y (2008) Classifying software changes: clean or buggy? *IEEE Trans Softw Eng* 34(2):181–196
- Knight JC, Strunk EA, Sullivan KJ (2003) Towards a rigorous definition of information system survivability. Paper presented at the DARPA information survivability conference and exposition, Washington, DC, 22–24 Apr 2003
- Kotonya G, Sommerville I (1998) Requirements engineering: processes and techniques, 1st edn. Wiley, New York, NY
- Kruchten P, Lago P, van Vliet H (2006) Building up and reasoning about architectural knowledge. Paper presented at the international conference on the quality of software architectures, Västerås, 27–29 June 2006
- Lago P, Avgeriou P (2006) First workshop on sharing and reusing architectural knowledge. *SIGSOFT Softw Eng Notes* 31(5):32–36
- Lam W, Shankaraman V (1998) Managing change in software development using a process improvement approach. Paper presented at the 24th annual Euromicro conference, Västerås, 27 Aug 1998
- Lehmann MM, Perry DE, Ramil JF (1998) Implications of evolution metrics on software maintenance. Paper presented at the international conference on software maintenance, Bethesda, MD, 16–19 Nov 1998
- Lund AM (2001) Measuring usability with the USE questionnaire. *Usability Interface* 8(2):3–6
- Majidi E, Alemi M, Rashidi H (2010) Software architecture: a survey and classification. Paper presented at the 2nd international conference on communication software and networks, Singapore, 26–28 Feb 2010
- Medvidovic N, Taylor RN (2010) Software architecture: foundations, theory, and practice. Paper presented at the 32nd ACM/IEEE international conference on software engineering, Cape Town, 1–8 May 2010
- Mesly O (2016) Project feasibility: tools for uncovering points of vulnerability. CRC Press, New York, NY
- Moore M, Kaman R, Klein M, Asundi J (2003) Quantifying the value of architecture design decisions: lessons from the field. Paper presented at the 25th international conference on software engineering, Portland, OR, 3–10 May 2003
- Mouratidis H, Sunyaev A, Jurjens J (2009) Secure information systems engineering: experiences and lessons learned from two health care projects. Paper presented at the international conference on advanced information systems engineering, Amsterdam, 8–12 June 2009
- Nielson J (2012) Usability 101: introduction to usability. <https://web.archive.org/web/20110408184029/http://www.useit.com/alertbox/20030825.html>. Accessed 5 May 2019
- O’Neal JS, Carver DL (2001) Analyzing the impact of changing requirements. Paper presented at the IEEE international conference on software maintenance, Florence, 7–10 Nov 2001

- O'Reilly C, Morrow P, Bustard D (2003) Lightweight prevention of architectural erosion. Paper presented at the 6th international workshop on principles of software evolution, Helsinki, 1–2 Sept 2003
- Pinto JK, Slevin DP (1988) Project success: definitions and measurement techniques. *Proj Manag J* 19(1):67–72
- Shenhar AJ, Dvir D, Guth W, Lechler T, Milosevic D, Patanakul P, Poli M, Stefanovic J (2007) Project strategy: the missing link. In: Shenhar AJ, Milosevic D, Dvir D, Thamhain H (eds) *Linking project management to business strategy*. Project Management Institute, Newtown Square, PA, pp 57–76
- Tang A, Babar MA, Gorton I, Han J (2006) A survey of architecture design rationale. *J Syst Softw* 79(12):1792–1804
- Tang A, Jin Y, Han J (2007) A rationale-based architecture model for design traceability and reasoning. *J Syst Softw* 80(6):918–934
- Tang A, Liang P, van Vliet H (2011) Software architecture documentation: the road ahead. Paper presented at the 9th working IEEE/IFIP conference on software architecture, Boulder, CO, 20–24 June 2011
- Taylor RN, Medvidovic N, Dashofy E (2009) *Software architecture: foundations, theory, and practice*. Wiley, Hoboken, NJ
- Tekinerdogan B (2004) ASAAM: aspectual software architecture analysis method. Paper presented at the 4th working IEEE/IFIP conference on software architecture, Oslo, 12–15 June 2004
- Thiebes S, Dehling T, Sunyaev A (2016) One size does not fit all: information security and information privacy for genomic cloud services. Paper presented at the 24th European conference on information systems (ECIS), Istanbul, 12–15 June 2016
- Tiwana A (2015) Evolutionary competition in platform ecosystems. *Inf Syst Res* 26(2):266–281
- Tiwana A, Konsynski B, Bush AA (2010) Platform evolution: coevolution of platform architecture, governance, and environmental dynamics. *Inf Syst Res* 21(4):675–687
- Toora S-u-R, Ogunlana SO (2010) Beyond the ‘iron triangle’: stakeholder perception of key performance indicators (KPIs) for large-scale public sector development projects. *Int J Proj Manag* 28(3):228–236
- Williams BJ, Carver JC (2010) Characterizing software architecture changes: a systematic review. *Inf Softw Technol* 52(1):31–51
- Wojcik R, Bachmann F, Bass L, Clements PC, Merson P, Nord R, Wood WG (2006) Attribute-driven design (ADD), version 2.0. <https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=8147>. Accessed 16 Sept 2019

Further Reading

- Albin S (2003) *The art of software architecture: design methods and techniques*. Wiley, New York, NY
- Bass L, Clements P, Kazman R (2012) *Software architecture in practice*. Addison-Wesley, London
- de Boer RC, Farenhorst R, Lago P, van Vliet H, Jansen AGJ (2007) Architectural knowledge: getting to the core. Paper presented at the 3rd international conference on the quality of software architectures, Medford, MA, 11–13 July 2007
- Gilb T (2005) *Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using planguage*. Elsevier, Butterworth-Heinemann, Oxford
- Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, America P (2007) A general model of software architecture design derived from five industrial approaches. *J Syst Softw* 80(1):106–126
- Taylor RN, Medvidovic N, Dashofy E (2009) *Software architecture: foundations, theory, and practice*. Wiley, Hoboken, NJ

Chapter 4

Internet Architectures



Abstract

In order to explain how the Internet works, this chapter takes a closer look at the architecture that underlies the Internet, as well as at its architectural principles and mechanisms. After providing a brief overview of the Internet's history, this chapter examines today's core infrastructure and explains the role of Internet service providers. In addition, the essential mechanisms enabling Internet communication are explained, namely the Internet Protocol (IP) suite, IP addresses, the domain name system (DNS), as well as IP packet routing and forwarding. This chapter also explains how large content providers, like Google, Amazon, and Netflix, provide Internet users all over the world with efficient and reliable services by utilizing specialized content delivery networks. The description of four emerging architectural concepts that extend the established Internet architecture with more efficient and/or effective ways of providing innovative Internet services (i.e., software-defined networking, network function virtualization, overlay networks, and information-centric networking) conclude this chapter.

Learning Objectives of this Chapter

The main learning objective of this chapter is to understand the structure and the inner workings of today's Internet architecture. After studying this chapter, readers will understand the key organizations that keep the Internet running and how these players interact with one another. From a technical perspective, readers will learn how messages are exchanged by means of the Internet's network infrastructure, which includes understanding important Internet protocols, how domain names function, and messages' routing between their senders and receivers. Readers will also learn about the mechanisms that help deliver content simultaneously and efficiently to a large number of Internet users. Finally, readers will understand what allows the Internet to evolve continuously and the emergent architectural concepts that fuel this process.

Structure of this Chapter

This chapter about Internet architectures is structured as follows: The first two sections introduce the fundamental Internet concepts, including a brief overview of the most important milestones in the Internet's history and an introduction to the

structure and governance of the Internet's network infrastructure. The second section covers the IP suite, which enables communication between Internet participants. Consequently, the IP suite's individual components are introduced, their interactions explained, and IP addresses' structure and purpose are introduced. Building on this, the more advanced concepts, namely domain name resolution and IP routing, are explained. The third section explains how content delivery networks work, the different types, and how they differ from one another. Concluding this chapter, the last section sheds light on four emergent Internet network architecture concepts.

4.1 History of the Internet

The origins of today's Internet go back to the U.S. Department of Defense's intention to ensure communication in the aftermath of a conventional or even nuclear strike during the Cold War. The Soviet Union's launch of its first orbiting satellite, Sputnik, in 1957 aroused the U.S. military's concerns about attacks from space. At that time, the national defense network relied on the public telephone system, which a single orbital strike could easily disrupt. To mitigate this and similar national security threats, the Advanced Research Projects Agency (ARPA), today known as the Defense Advanced Research Projects Agency, was founded on February 7, 1958. In 1962, an ARP scientist (J.C.R. Licklider) developed a first concept for connecting multiple computer systems to form a long-distance communications network that would keep operating even in the event of a nuclear attack. In 1965 the invention of packet switching technology, which allowed for reliable data transmission and provided an early form of routing devices (i.e., interface message processors), complemented the computer network idea. Based on these key technology advancements, the ARPA completed the first version of the ARPANET in 1969. On November 21, 1969 the first link was established between the University of California, Santa Barbara's and the Stanford Research Institute's computer systems via the ARPANET. Less than two weeks later, on December 5, 1969, another two nodes joined the network (University of California, Los Angeles and the University of Utah).

In 1971, Ray Tomlinson sent the first email between two computers over the ARPANET. Two years later, the ARPANET expanded internationally by connecting the University College of London (England) and the Royal Radar Establishment (Norway). In 1974, Vinton Cerf and Robert Kahn, later referred to as the fathers of the Internet, published "A Protocol for Packet Network Interconnection," (Cerf and Kahn 1974) detailing the design of the Transmission Control Protocol (TCP), an important part of today's Internet protocol stack (see section 4.3). The first attested use of the term Internet – an abbreviation for internetworking (i.e., the interconnection of two heterogeneous networks) – is found in the TCP's formal specification (Cerf and Sunshine 1974).

In 1981, the National Science Foundation (NSF) indirectly opened up the ARPANET to computer science institutions by creating the Computer Science Network (CSNET), which eventually comprised more than 180 institutions (Stewart 2000). While not all institutions could be directly connected to the ARPANET due to technical and organizational limitations, the few universities that were part of both ARPANET and CSNET acted as gateways to the networks. Two years later, in 1983, the ARPANET was restructured and divided into a highly secured military network (MILnet) and an open civil network (ARPANET), which remained connected by means of controlled gateways. In the same year, the ARPANET officially adopted the open networking protocols TCP and the Internet Protocol (IP), which, combined, are called TCP/IP. Owing to TCP/IP's simplicity and scalability, it removed the existing capacity constraints and allowed almost any network to be connected to the ARPANET, irrespective of their local characteristics (Kozierok 2005; Fall and Stevens 2011). Based on this technology, the NSF created a new network, the NSFNet, in 1984 to connect to its supercomputer centers located at various U.S. universities (Frazer et al. 1996). Soon after its launch, numerous smaller university networks across the U.S. and the rest of the world connected to the NSFNet, including the ARPANET. Owing to its rapid growth, the NSFNet took on the role of the U.S. Internet backbone¹, which interconnected the different regional networks to the one large network, today referred to as the Internet, in 1988 (Mills and Braun 1987). As just one of many Internet subnets, the ARPANET started losing its relevance, leading to its decommission in 1990.

In 1989, Tim Berners-Lee created the hypertext transfer protocol (HTTP) at the European Organization for Nuclear Research, which in turn led to the development of the first Web browser called the *WorldWideWeb*. This was the birth of the World Wide Web (WWW), which, from today's perspective, indicates a far broader term than just a Web browser. Finally, in 1995, the NSFNet modified its acceptable use policy in terms of applying the network for commercial use and, soon after, it was gradually replaced by backbones operated by commercial Internet service providers (ISPs). At the same time, ISPs, like Compuserve and America Online, also started building network access points that allowed the Internet's commercialization and privatization.

The rest is history in the making. Over the past 60 years, the Internet has evolved from a simple research concept to a technology which more than four billion people currently use and which continues to evolve through the development of novel ideas, services, and architectures.

¹A backbone network establishes interconnections between two or more separate networks by establishing a central data path between them (Piliouras and Terplan 1998).

4.2 Today's Internet Network Infrastructure

The Internet is a large network of computer networks that connects billions of computing devices around the globe. Each of these devices can communicate with any other computing device as both are connected to the Internet and use the same communication protocols.

Computer Networks

Understanding the concept of computer networks is paramount to understanding the Internet's architecture as we know it. A computer network is a collection of interconnected devices for electronic data communication (NIST 2013; Mansfield and Antonakos 2009). In terms of its physical structure, a computer network consists of linked network devices (i.e., nodes) that either serve as endpoints (e.g., servers, personal computers, or smartphones) or network devices that facilitate the relaying of data between endpoints (e.g., modems, hubs, bridges and switches). Network nodes rely on standardized communication protocols to exchange information and to transport data to their intended endpoint. Such communication protocols define the way in which data should be sent and received precisely. Modern Using either wired or wireless technologies computer networks connected to the Internet are often based on the TCP/IP protocol suite, which will be explained in detail in section 4.3.

Computer Network

A computer network is a collection of computers and devices connected for the purpose of electronic data communication that allows them to share information and services (Mansfield and Antonakos 2009; NIST 2013).

Computer networks may be private or public. *Private networks* allow only authorized nodes to access the network and communicate with other connected nodes. Depending on the particular network configuration, a node can identify itself by means of credentials (e.g., username/passwords or certifications) or is granted access by a network administrator. In contrast, with no or only a few exceptions, any node can access *public networks*. While the resulting openness facilitates communication between nodes, it also makes it easy for malicious nodes to join the network, which increases the security risks compared to those of private networks.

The geographical distance between individual nodes also categorizes computer networks. A local area network (LAN) connects nodes in close proximity, for instance, within the same room, on the same floor, or in the same building. Nodes that are further apart, but still confined to a 50 km radius, are linked by a metropolitan area network (MAN) (Gokhale 2005). If the distance between nodes exceeds 50 km, they are interconnected via a wide area network (WAN), which is usually not restricted to an enclosed geographical space. As shown in Table 4.1, longer distances between nodes comes at the cost of lower transmission rates and higher propagation delays due to the underlying transmission medium's physical limitations. Using

either wired or wireless technologies (or a mixture of both) allows for establishing a connection between individual network nodes. A wired network connection uses transmission media like copper or optical fiber cables. Wireless networks can be based on a variety of transmission methods, like radio, microwaves, or optical signals (e.g., lasers).

Table 4.1 Overview of Computer Network Types

	Wide Area Network	Metropolitan Area Network	Local Area Network
Abbreviation	WAN	MAN	LAN
Purpose	Connects computing devices over a large geographical distance or even those across the globe	Interconnects computing devices within a city or metropolitan area	Connects computing devices within the same room or building
Geographic Expanse	More than 50 km	5 to 50 km	Less than 5 km
Transmission Rates	Low	Medium	High
Propagation Delay	High	Medium	Low
Applications	Networks between globally distributed enterprise branches, the Internet	City networks, several industrial facilities in close proximity	Industrial plants, business offices, university campuses, domestic homes

The Internet – A Network of Networks

Given the description of computer networks, the Internet can be defined as a public wide area network based on the TCP/IP protocol suite to interconnect computer systems across the world. Nodes connected to the Internet provide a large variety of information resources and enable services, such as e-mail, voice over IP, peer-to-peer file sharing, and, of course, the meta-service WWW. It is important to emphasize that the terms Internet and WWW are not synonymous. The WWW, or simply the ‘Web,’ is just one service that the Internet provides. The WWW is basically a set of resources (e.g., webpages, documents, and images) connected by hyperlinks and accessed via Uniform Resource Identifiers (URIs) (Berners-Lee et al. 2004). Further, while the well-known Hypertext Transfer Protocol (HTTP) and its extension, the Hypertext Transfer Protocol Secure (HTTPS) are essential for data exchange via the WWW, they are only two of various protocols used for communication on the Internet.

World Wide Web

The World Wide Web (WWW, or simply the Web) is an information space (on the Internet) in which global identifiers called Uniform Resource Identifiers identify the items of interest, referred to as resources (Berners-Lee et al. 2004).

At the beginning of this chapter, the Internet was described as a network of computer networks. Even though, from users' perspective, the Internet does seem like one large network, its fragmented structure becomes obvious when taking a closer look at its actual topology. Fig. 4.1 presents a simplified depiction, in which the Internet is a hierarchically structured network of numerous WANs.

The *global Internet backbone*, a network that links independent WANs from different areas of the world, known as Internet regions, resides at the highest hierarchy level (or tier) (Norton 2011). These WANs are usually operated by large national telecommunication companies referred to as tier 1 ISPs (i.e., Internet Service Providers) or Backbone Internet Providers. Tier 1 ISPs, like AT&T, CenturyLink, and Deutsche Telekom, use the global Internet backbone to exchange data traffic across Internet regions. This means that if a node *A* located in one Internet region sends data to a node *B* in another Internet region, the transmission will, at some point, cross the border between the regions by being passed from a tier 1 ISP responsible for *A*'s internet region to a tier 1 ISP responsible for *B*'s internet region. This exchange of data traffic between two ISPs is called *peering*. In most cases, peering between tier 1 ISPs is carried out over high-speed optical fiber cables that either interconnect their networks (private peering) directly, or are brought together for public peering in an Internet Exchange Point (IXP), which is a provider-neutral data center where several ISPs connect with one another to exchange data traffic.

Since peering across Internet regions has high infrastructure requirements (e.g., submarine cables or communication satellites) and high demands regarding the provider's organizational and technical capabilities, the vast majority of today's ISPs reside on tier 2 and 3 of the Internet's topology. These lower-tier ISPs often act as middlemen between their end-customers (e.g., businesses and consumers) and other ISPs by relaying Internet traffic within the same Internet region.

To this end, tier 2 ISPs (regional ISPs) operate their own networks. If two communicating nodes are connected to the same tier 2 network, the relevant ISP delivers data transmissions between them directly without ever leaving its network. If two nodes are connected to different ISP networks in the same Internet region, the relevant two ISPs will engage in peering, either directly or indirectly, over additional ISP networks on the path between the nodes. Similar to cross-regional peering, peering within an Internet region is either handled privately (i.e., directly) between two ISPs, or over a Metropolitan Area Exchange (MAE), which is a regional version of an IXP operated in densely populated areas. In order to relay data to and from other Internet regions, tier 2 ISPs need to engage in peering with a tier 1 ISP in their region. This process may involve fees for the lower tier ISP. Although ISPs on the same tier usually do not charge one another for transit, upstream peering from lower to higher tier ISPs creates fees for the lower tier ISP if no other peering agreements was closed.

Tier 3 ISPs (local ISPs) only operate by purchasing data transit from tier 1 and 2 providers, and do not engage in peering between one another. Tier 3 ISPs are specialized in connecting residential homes and businesses with the Internet. However, the boundaries between tier 2 and tier 3 ISPs are blurred and the terms are sometimes used interchangeably due to their similar proximity to the end-customer

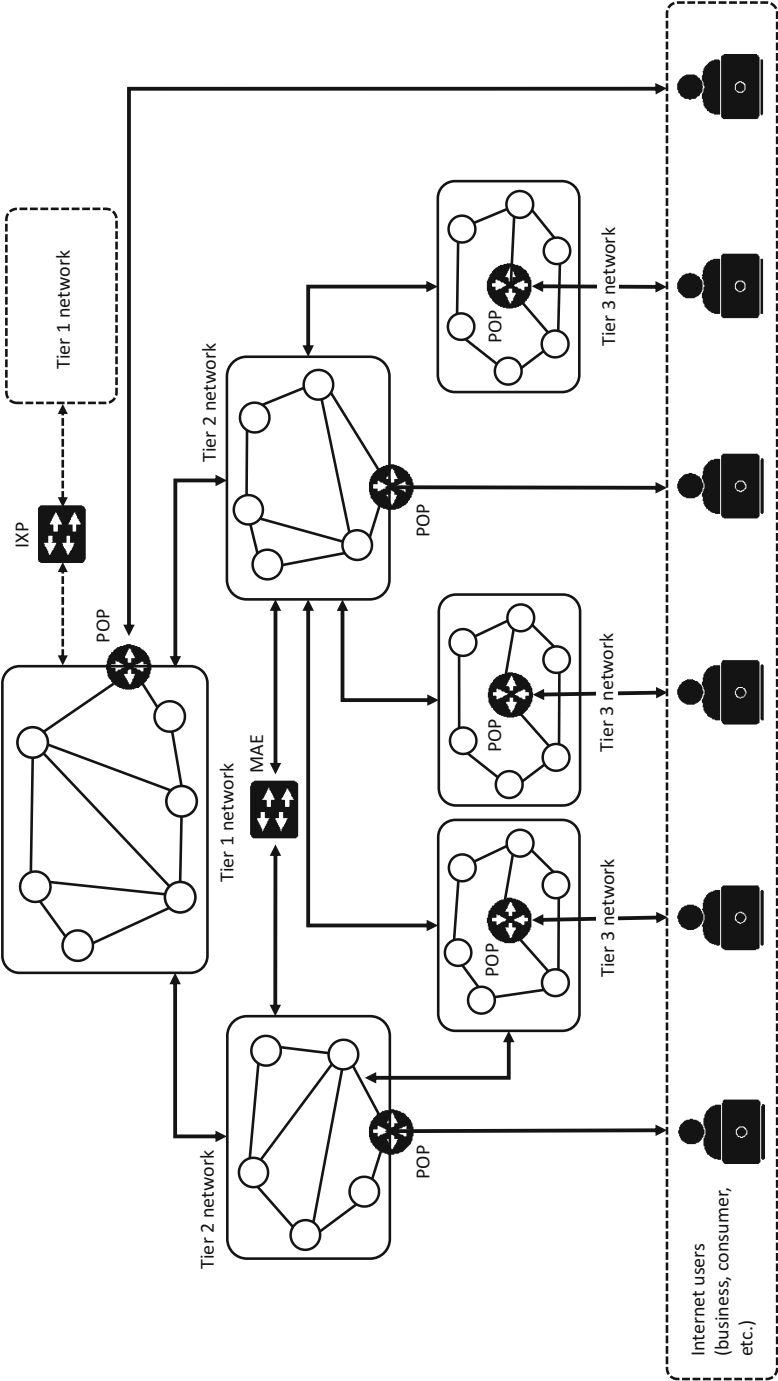


Fig. 4.1 Hierarchical Structure of the Internet

business. Furthermore, some organizations operating large networks (e.g., universities, global enterprises, and government institutions) often bypass tier 2 and 3 ISPs by directly engaging themselves in peering and transit purchasing.

Internet users may be ISP customers on any of the Internet's topology's tiers. In order to access their ISP's services, customers' nodes need to establish a connection to the ISP's network. This is done over either a wired (e.g., telephone system) or wireless (e.g., satellite link or cell phone network) connection that ends at the ISP's nearest physical access point. Such access points, or points of presence (POPs), are often located within the facility of the telecommunications provider responsible for the physical communication infrastructure to the customer. Finally, the POPs are connected to the relevant ISP's network, which, as described above, provides direct or indirect access to the global Internet backbone.

Fig. 4.2 provides a simplified depiction of how two nodes in different Internet regions are connected over ISP networks. An Internet user, whose computer is connected via a landline to the POP of her tier 2 ISP, initiates the transmission in the example. The tier 2 ISP detects that the transmission's destination is not within its network, nor in the same Internet region, therefore relaying it to the relevant tier 1 ISP. The tier 1 ISP then engages in peering over an IXP with a tier 1 ISP responsible for the destination Internet region. Within the destination region, the transmission is again relayed to the receiving node's tier 2 ISP. A local POP establishes the connection between the tier 2 ISP and the destination endpoint, finally delivering the transmission.

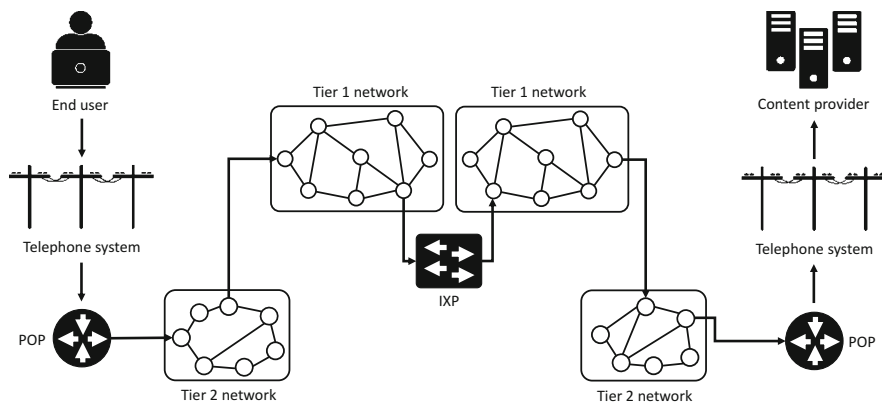


Fig. 4.2 Internet Connection Example

Regulation of the Internet

Several organizations standardize, manage, and advance the various facets of the Internet. Table 4.2 provides a list of the most important organizations, which are detailed in the remainder of this section.

The Internet Society (ISOC) takes a central role in the Internet's development and technical evolution. The ISOC is a nonprofit organization providing consulting for Internet-related public policy, training for individuals, and serves as an umbrella organization for the groups responsible for Internet standards and technologies, including the Internet Engineering Task Force (IETF), the Internet Research Task Force (IRTF), and the Internet Architecture Board (IAB). The IAB is a technical advisory board entrusted with the task of overseeing the architectural development of the Internet, which includes supervising the IRTF and IETF (Chapin 1992). The IRTF researches future Internet technologies to solve basic technical problems and drives the Internet's long-term evolution (Falk 2009). When new technologies are ready for implementation, they are passed on to the IETF, which is an open standards organization tasked with maintaining and developing Internet technology further. The IETF is particularly concerned with the standardization of the communication protocols used on the Internet, such as the IP, User Datagram Protocol (UDP), TCP, Stream Control Transmission Protocol (SCTP), and HTTP.

The Internet Corporation for Assigned Names and Numbers (ICANN) is a non-profit corporation whose primary role is to oversee the IP address and domain name space. The ICANN delegates the administration and registration of IP addresses and address spaces to five Regional Internet Registries, namely the African Network Information Center (AFRINIC), the American Registry for Internet Numbers (ARIN), the Asia-Pacific Network Information Centre (APNIC), the Latin America and Caribbean Network Information Centre (LACNIC), and the Réseaux IP Européens Network Coordination Centre (RIPE NCC). In addition, the ICANN oversees and manages the domain name system (DNS; see section 4.3.3). All domain names used on the Internet, such as [google.com](https://www.google.com) and [amazon.com](https://www.amazon.com), are assigned by means of a registration process that local registrars accredited by the ICANN undertake. The ICANN is also responsible for introducing new top-level generic domains, like .app and .news.

The *World Wide Web Consortium* (W3C) is responsible for developing interoperable technologies (specifications, guidelines, software, and tools) for the WWW. The W3C's primary purpose is to ensure the WWW is as accessible as possible to "all people, whatever their hardware, software, network infrastructure, native language, culture, geographical location, or physical or mental ability." (W3C 2017) To this end, the W3C maintains well-known Web technology standards, for instance, the Hypertext Markup Language (HTML), the Extensible Markup Language (XML), and Cascading Style Sheets (CSS).

Table 4.2 Organizations for Standardization and Regulation of the Internet (selection)

Organization	Main purpose
The Internet Society (ISOC)	Provides the organizational structure that supports the process of Internet standard development
Internet Research Task Force (IRTF)	Researches solutions for basic technical problems and the Internet's long-term evolution
Internet Engineering Task Force (IETF)	Develops and maintains Internet technology standards (e.g., TCP/IP)
Internet Architecture Board (IAB)	Architectural oversight of the Internet development (e.g., oversight of IETF and IRTF)
Internet Corporation for Assigned Names and Numbers (ICANN)	IP address space allocation and management of the domain name system
Regional Internet Registries (RIR)	Administration and registration of IP address spaces for individual Internet regions (e.g., RIR for Canada and the United States: American Registry for Internet Numbers ARIN)
World Wide Web Consortium (W3C)	Develops standards, guidelines, and software for the WWW (e.g., Hypertext Markup Language, Extensible Markup Language, Cascading Style Sheets, and SOAP)

4.3 The Internet Protocol

4.3.1 Internet Protocol Suite

The Internet Protocol Suite consists of a set of protocols enabling communication between network nodes both on the Internet and over other computer networks (Baker 2009). It is also known as the TCP/IP protocol suite (or in short, TCP/IP) after two of its most important protocols: TCP and IP. TCP/IP provides end-to-end data communication over heterogeneous physical networks. More specifically, the protocol suite defines how data should be packaged, addressed, sent, routed, and received. The IETF maintains the technical specifications of many of TCP/IP's constituent protocols.

The TCP/IP protocol suite's independence from specific hardware and software platforms is one of its essential qualities. As long as hardware and software layers are capable of transmitting and receiving data via a computer network, TCP/IP can be used to establish communication between the nodes on that network. The platform independence also facilitates communication between nodes on different interconnected networks, which, given the Internet's composite network architecture, is a highly important feature for Internet communication. Another key TCP/IP quality is its robustness. Owing to the Internet's military origins, one of the fundamental requirements for its protocols was their ability to continue operating even if large parts of the network suddenly failed. TCP/IP was therefore designed with built-in failure recovery mechanisms to provide reliable end-to-end communication even under adverse conditions. For instance, if data is (partially) lost during transmission,

TCP/IP automatically resends the affected data (parts), but not necessarily taking the same network route on which the original data transmission went missing.

Internet Protocol Suite

The Internet protocol suite is a set of protocols that enables Internet communication by specifying data transmission, addressing, and routing (Baker 2009).

TCP/IP is hierarchically structured into four layers stacked on top of one another (i.e., a protocol stack), namely: the (1) data-link layer, (2) network layer, (3) transport layer, and (4) application layer (IETF 1989b, a). As depicted in Fig. 4.3, each of the four layers comprises different protocols. The upper layers of the protocol suite are logically closer to the software applications with which users interact (e.g., Internet browser), and the lower layers are closer to the physical networking hardware. Communication between the layers is established by standardized interfaces between adjacent layers, which hide the actual implementation of a layer's functionality. Each layer offers services to the layers immediately above and uses the services of the layer directly underneath. This strict separation creates flexibility and greater independence from the services' implementation in terms of both hardware and software. How these services are implemented (i.e., which protocols are used), do not concern the layers using them. This means that, on the one hand, layers at the top of TCP/IP (i.e., application and transport layer) do not know how the networking hardware handles individual bits of a data transmission. On the other hand, the lower layers (i.e., network and data-link layer) distinguish between the different data formats and protocols that particular applications require. The four abstraction layers' purpose and scope are subsequently described in detail.

The **application layer** provides applications with standardized interfaces that allow them to send data to other applications or receive data from them via a network. This allows different applications to communicate with one another as long as they use the same application layer protocols. Since the application layer protocols are very near the actual applications, they support particular operational scenarios instead of general networking functions. These types of protocols therefore use the networking services that the underlying lower layers offer, specifically the transport layer. The application layer protocols regard the protocol layers below them as black boxes offering a stable network connection for communication between applications, but also do not affect the layers below. This means that, for instance, network devices operating on lower-level protocols, like routers and switches, do not generally examine the encapsulated data transmissions. Instead, these devices are more concerned with relaying them. Consequently, all application-near protocols can be found on the application layer, including the HTTP, File Transfer Protocol (FTP), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP), and Simple Network Management Protocol (SNMP).

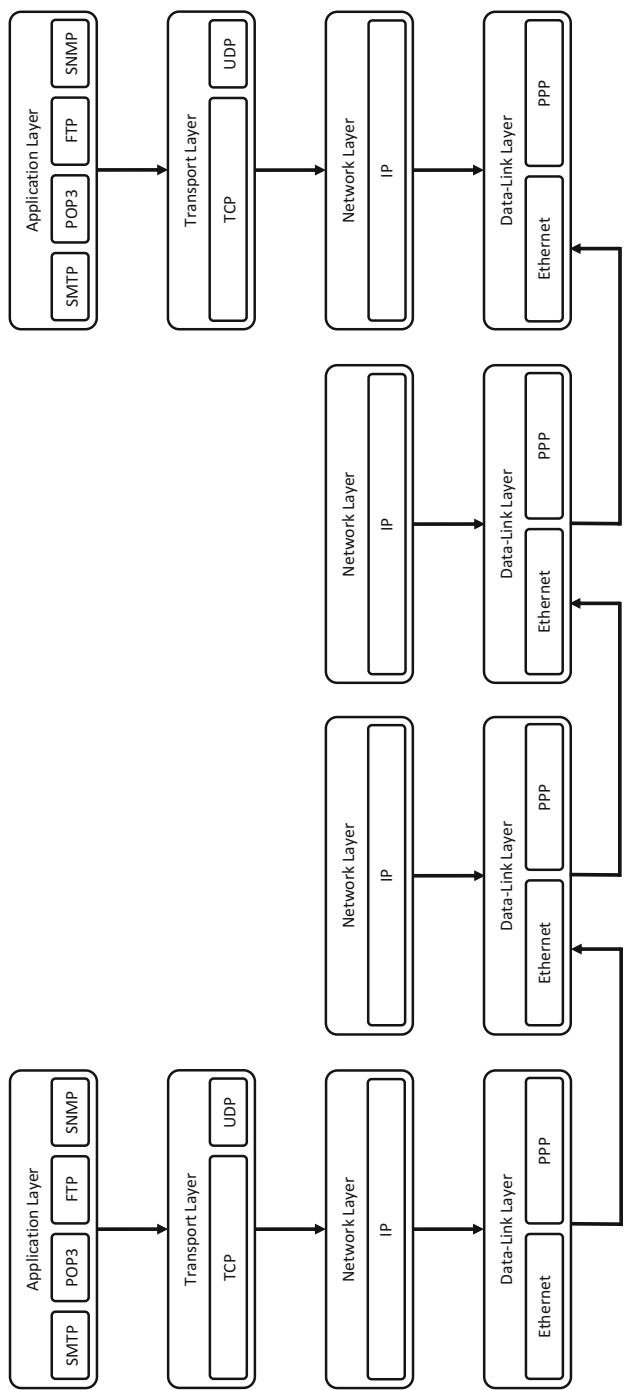


Fig. 4.3 The Four Layers of the Internet Protocol Suite

The **transport layer** is responsible for the correct transfer of data between network nodes, independent of their application, specific data structures, and underlying network (Hunt 2002). At this layer, the most frequently used protocol is the TCP, whose primary function is providing connection-oriented, reliable data delivery. The TCP ensures that, for instance, data arrive in the order they were sent, that they arrive correctly, that duplicate data are discarded, and that data lost in transit are resent. In order to do so, the TCP establishes a bidirectional connection over which the sending node transmits data to the receiving node, which confirms its receipt with messages that acknowledge receipt of the data. The TCP is therefore well suited for cases where data integrity is important. However, this level of integrity comes at the cost of efficiency due to the decreased throughput. If throughput is more important than the data integrity, an alternative transport layer protocol, the UDP, can be used. As a connectionless protocol, the UDP neither checks that a data transmission actually arrived at the receiving node, nor that such a node even exists. The UDP also doesn't provide any error-recovery mechanisms, because, for example, corrupted or lost data transmissions are not resent. By combining connectionless data transmissions and no error-recovery, the UDP allows for transmitting data at far lower overhead costs and less communication delay between the hosts. The UDP's use is therefore primarily in applications where large amounts of data need to be transmitted with little delay and where the loss of a few data bits can be tolerated, such as with video and audio streaming. Nonetheless, applications utilizing the UDP might, if required, also implement integrity mechanisms on the application protocol level.

The **network layer** (or Internet layer) is responsible for transporting data between the right nodes within a network or across multiple networks. The network layer has two important functions, namely node addressing and data routing. To this end, the IP, which is the most important protocol on this layer, specifies an addressing method that unambiguously identifies a data transmission's sending and receiving node: the IP address. As will be explained in detail in section 4.3.2, IP addresses are numerical labels uniquely assigned to each node within a network. Based on this addressing system, the IP also provides routing functions that forwards data to a specific destination in the network, identified by its unique IP address. However, the IP is a connectionless protocol that not only does not expect reliability from the lower layers (e.g., data-link layer), but also does not itself provide mechanisms contributing to reliable data delivery, like error recovery. If required, a higher-level protocol, like the TCP, needs to provide these functions. Other network layer protocols are the Internet Group Management Protocol (IGMP), Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP), and Reverse Address Resolution Protocol (RARP).

The **data-link layer** (also network interface layer or physical layer) provides an interface to the actual physical networking hardware that links two or more distributed nodes. This interface allows for sending or receiving data transmissions to or from a data-link interface of a different network node on the same network segment (link). The data-link layer represents the least abstract specification of the interactions with the hardware components. However, since TCP/IP is designed to be independent of specific hardware implementations (i.e., can be implemented regardless of the underlying network technology), the data-link layer does not decree the

use of particular protocols. Instead, the data-link layer provides a standardize, but flexible, interface for accessing such protocols, which, for instance, describe the structure of the underlying network, or specify the interfaces that allow for forwarding a data transmission over wired or wireless communication technology to a neighboring network node. The following protocols are common on this layer: IEEE 802.2, IEEE 802.11, Ethernet, and Fiber Distributed Data Interface (FDDI).

To illustrate the individual layers' collaboration, Fig. 4.4 shows a simplified process of retrieving a webpage from a Web server using TCP/IP. In the example, a user opens the webpage with the URI <http://example.com/products.html>, or, in other words, the user uses a Web browser application to communicate with a Web server. To do so, the user typed the URI into the browser application, which runs on the application layer. When the enter key is pressed, the browser program sends a message to the transport layer “Get: example.com/products.html.” At this layer, the TCP adds information to the message, including the network port specifying the service provided by the receiving node for which the transmission is meant (e.g., port 80 on a Web server) and an error checksum that allows for checking the message's integrity (i.e., that its content has remained unchanged) at its destination. The TCP then passes the message plus the extra information to the network layer where the IP adds information required for delivery to the specific destination node, including the IP addresses of the source and destination. The TCP had determined the destination IP address earlier by looking up the domain name address (see DNS later), example.com (see section 4.3.3). The data transmission is then sent to the Web server via the Internet by utilizing the network hardware to which the data-link layer at the bottom of the TCP/IP stack provides an interface. At the receiving end (the server), the data transmission crosses the server's protocol stack, but in reverse order, by starting at the data-link layer. Once the message reaches the application layer, the server software identifies the requested HTML document *products.html* and prepares a response message that includes the document. The response message follows the above described path back to the user's browser program, which will display the HTML file.

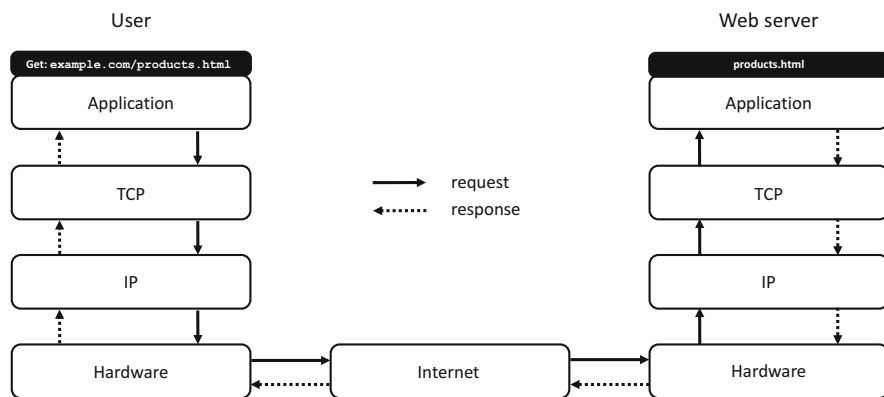


Fig. 4.4 TCP/IP Transmission Example

4.3.2 IP Addresses

As described in the previous section, TCP/IP enables communication between the hosts located on the same network, or on different interconnected networks. In order to ensure that the data can be routed to the target host, each endpoint requires a unique address, called the IP address (Stevenson and Waite 2011). RIRs assign IP addresses centrally, but are in turn supervised by the ICANN to ensure that these addresses are unique throughout the Internet. Currently, two different versions of the IP are in use: IPv4 and IPv6. The IP version 4 (IPv4) had its first large-scale application in 1983 as part of the ARPANET. An IPv4 address is a 32-bit-long binary number, allowing for specifying 2^{32} (4,294,967,296) unique addresses. Owing to the Internet's rapid growth, this pool of available addresses soon became too small, prompting the IETF to increase the address space in 1995 by evolving IPv6 into the IP version 6 (IPv6) (Deering and Hinden 1995, 1998, 2017). IPv6 increased the address size to 128 bits, which provides up to 2^{128} (approximately 3.403×10^{38}) addresses, which are deemed ample for the foreseeable future. Currently, both IP versions are used in parallel. Beside other technical changes, IPv6's extension of the address space also resulted in a different address format, as will be shown in the remainder of this section.

IP Address

An IP address is a group of binary numbers uniquely identifying a node within a network that uses the Internet Protocol (Stevenson and Waite 2011).

As depicted in Fig. 4.5, IPv4 addresses consist of four 8-bit-long blocks adding up to 32 bits. In order to increase their readability, IPv4 addresses are often split into four 1-byte-long segments (i.e., octets) converted into corresponding decimal numbers between 0 and 255 and separated by dots. An IPv4 address can also not only identify a particular node on the Internet, but also the network to which it is directly connected. To this end, an IPv4 address can be logically split into two segments: a network identifier and a host identifier. The network identifier, which identifies a node's network, can be one to three octets long, depending on the size of the network. The network identifier segment always starts from the left. The remaining segment of the network identifier specifies the node within the network (i.e., the host identifier). A *subnet mask* can be used to specify where an IP address's network identifier ends and the host identifier starts. The subnet mask is the length (number of bits) of the network identifier segment (i.e., the prefix length). This number, preceded by a slash, is appended at the end of an IPv4 address. An example of an IPv4 address whose first 24 bits (three octets) describe its subnet: 192.168.35.22/24. The remaining 8 bits (one octet) can be used within the network to address a maximum of 2^8 (i.e., 255) unique nodes.

Owing to IPv6 addresses' quadruple length compared to IPv4, their 128 bits of are normally depicted as a group of eight 16-bit hexadecimal number blocks (i.e., hexets) separated by colons. Each hexet consists of four hexadecimal digits

(bits) representing 16 binary digits, which halves the number of required blocks compared to the decimal notation. A typical IPv6 address: `3ffe:0db8:85a3:0000:0000:8a2e:0370:67cf`. IPv6 addresses can be further simplified by omitting the leading zeroes within a group, meaning that ‘0370’ can be shortened to ‘370,’ but not to ‘37.’ Two or more consecutive groups of all zeros can be replaced by two colons (::), as depicted in Fig. 4.5. When the address is expanded to its full 128-bit length, the double colon is replaced with groups of zeros until the address is again made up of 8 groups. However, if an address were to have multiple double colons, it would be impossible to determine where to add a specific number of groups to restore the address’s original representation. Consequently, a valid IPv6 address is only allowed to contain a double colon once. In all other cases, a group must retain at least one hexadecimal digit to prevent ambiguities (Hinden and Deering 2006). By combining all the shortening rules, the above example address can also be written as: `3ffe:db8:85a3::8a2e:370:67cf`. Like IPv4, an IPv6 address can be separated into a network and host identifier by using a subnet mask that specifies a prefix length by using the above described slash notation. The most common prefixes used with IPv6 are multiples of four. For instance, the IETF defines /64 as a standard size for an IPv6 subnet, which allows for assigning 2^{64} unique addresses within the network. A /56 subnet would allow for 256 subnets of a /64 subnet size, and so on.

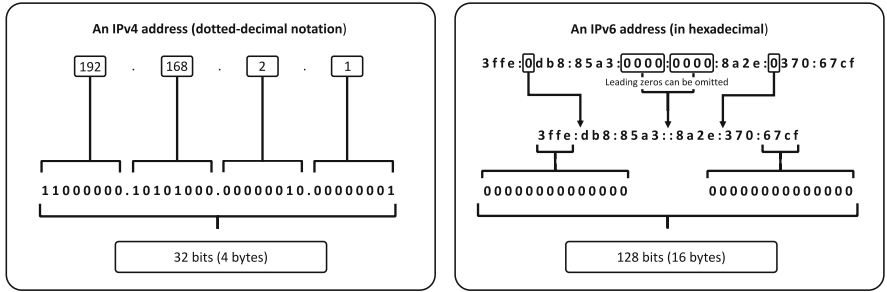


Fig. 4.5 Structure of IPv4 and IPv6 Addresses

4.3.3 Domain Name System

While IP addresses are an effective approach to communicate with a specific host, they are neither easy to memorize nor descriptive in terms of the network resource to which they are assigned. They are especially not practical for user-focused Internet services like email or the WWW. In 1983, the IETF therefore proposed the Domain Name System (DNS), which became one of the essential components of today’s Internet infrastructure (Mockapetris 1983a, b). DNS is a hierarchically structured, decentralized set of databases that associates *domain names* with a specific Internet resource or with collections thereof, such as computers, users, networks, and services. The DNS’s most important purpose is to translate memorable text-based domain names to numerical IP addresses, a process called DNS look-up. The DNS

allows Internet users to visit a website by typing in the domain name rather than its IP address. For instance, the domain name *google.com* in the address bar of a Web browser or Google’s Web server IP address *216.58.213.174* provide access to the website of Google’s Web search engine. The DNS also simplifies email by translating the domain name that follows the @ symbol into the respective mail server’s IP address.

Domain Name System (DNS)
The Domain Name System (DNS) is a hierarchically structured, distributed set of databases that maps IP addresses to corresponding domain names (Chandramouli and Rose 2006).

A domain name comprises one or more parts, called labels, joined by dots. As depicted in Fig. 4.6, these labels are organized hierarchically following the DNS’s tree-like structure. The highest level of this hierarchy (the root) has no DNS name. The top-level domains, including generic top-level domains, such as com, info, net, edu, and org, or country code top-level domains, like de, fr, or ca, are specified directly below the root. The second-level domains, which organizations and individuals (e.g., springer, example, google, amazon) can reserve, lie underneath the top-level domains. Finally, a domain name may specify a third-level domain, sometimes referred to as a subdomain or host name. Third-level domains are not mandatory and are generally used to specify a certain node inside an organization. The most common third-level domain is www, which usually links to a Web server providing WWW services. Each level adds a label to the domain name, which descends from the right to the left. For example, the domain name *www.example.com* comprises the *com* top-level domain, the second-level domain *example*, and the subdomain *www*.

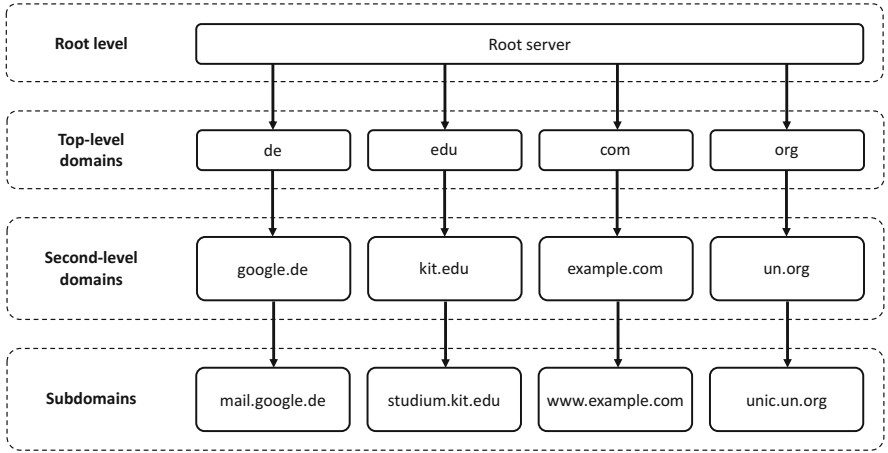


Fig. 4.6 DNS Domain Hierarchy

The DNS itself is basically a distributed network of individual DNS servers managing domain names and their associated records, which specifically include the aforementioned mapping between the domain names and IP addresses. These records are kept in DNS translation tables distributed to and stored on numerous DNS servers around the world. For efficiency's sake, most of these DNS servers do not keep a complete replication of the entire DNS translation table. Instead, most of these servers are assigned a specific domain zone that correspond to the DNS domain hierarchy. In addition, DNS servers have some knowledge of the DNS servers below and above their domain level, which allows them to refer incoming look-up request to an appropriate server. As exemplified in Fig. 4.7, the DNS look-up process involves requests to multiple DNS servers on the different domain hierarchy levels (top-down) in order to translate a domain name into an IP address. When a user wants to visit a website by using its domain name, the browser will send a request to a known DNS server that the user's ISP often maintains. If the DNS server already knows the domain name, it will provide the relevant DNS record directly. If the DNS server does not keep the requested domain name's record, it will send the request to the DNS server responsible for the relevant top-level domain. This top-level DNS server knows the IP addresses of all second-level DNS servers directly below it in the hierarchy and will again refer the requesting DNS server down to relevant second-level DNS server. This process continues down the DNS hierarchy until either the record for the particular domain name is found or the process fails. A DNS lookup is declared failed if, during this process, no suitable lower-level DNS server is found, or the process exceeds a specified time limit.

Since the DNS lookup process creates additional overhead communication costs and longer response delays when requesting resources on the Internet, ISPs cache requested DNS records for a certain time. Once a domain name's record is cached on an ISP's DNS server, this serves answers all similar requests directly, instead of first undertaking a DNS lookup starting with the DNS root layer. However, because domain records are not static, their mapping to IP addresses can change, in which case cached records will point to the wrong or a non-existent server. Hence, DNS caches are frequently updated.

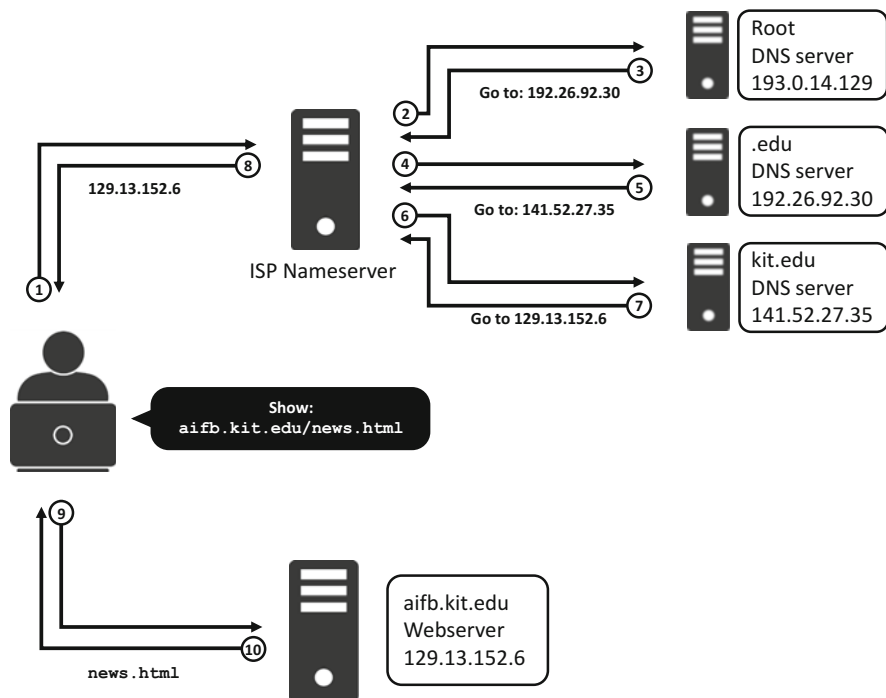


Fig. 4.7 Example DNS Lookup

4.3.4 IP-Routing and Packet Forwarding

In order to understand *how* data find their way through the Internet to their destination, it is important to first define *what* is transmitted. Up to this point in this chapter, any data sent via the Internet has been treated as a single coherent message. However, this is not actually how data are transmitted. TCP/IP uses a method called *packet switching*, which means that data are not transmitted as single units, but are divided into smaller blocks that are sent separately. These data blocks, called IP packets, are reassembled once they reach their destination. The Maximum Transmission Unit (MTU) size is one of the main reasons for using packet switching. The MTU defines the maximum data size of a packet sent or received in a single network transaction. The underlying network infrastructure, which makes a trade-off between different factors, such as efficiency and reliability, determines the MTU. For example, larger units allow for sending the same amount of data, but in fewer packets, thus increasing the efficiency by reducing the cumulative overhead costs. However, if a packet is corrupted in transit and has to be resent, a larger data unit will reduce the efficiency by increasing the overall amount of network traffic. A common MTU value of 1,500 bytes is common for Ethernet-based networks (Hornig 1984). This means, for example, that when downloading a document of 10 MB, the file cannot be

transmitted as a single data unit, as the package exceeds the MTU limitation of the underlying transmission system.

Packet Switching

Packet switching describes a switching and transmission technology which splits complete messages into smaller packets. These packets can be transmitted via a network’s different lines and the receiving host re-assembles them into the original message (Jordana 2002).

While a packet’s specific content varies in respect of the applied protocols, all packets generally consist of two parts: a header and a payload. The header portion stores overhead information on the packet’s content (e.g., the encoding style), the underlying protocol, and transmission-related data (e.g., the sending and receiving nodes’ IP addresses and a packet’s sequence number that specifies its reassemble order once all of a transmission’s packets have arrived at their destination). The payload comprises the actual data block being carried, which is the message content or a part of it. Like an onion, an IP packet usually comprises multiple layers, one for each protocol used for a particular connection, each of which has its own header. As depicted in Fig. 4.8, each layer has a payload and a header containing protocol information relevant for the layer. The next outer layer treats the entire layer to be encapsulated as payload (data) and adds its own header. In other words, each layer considers the packet it receives from the layer above it as one coherent data block. This process of attaching new headers to existing data is called *encapsulation* and, at the packet’s destination, the protocol stack reverses this process in the same order.

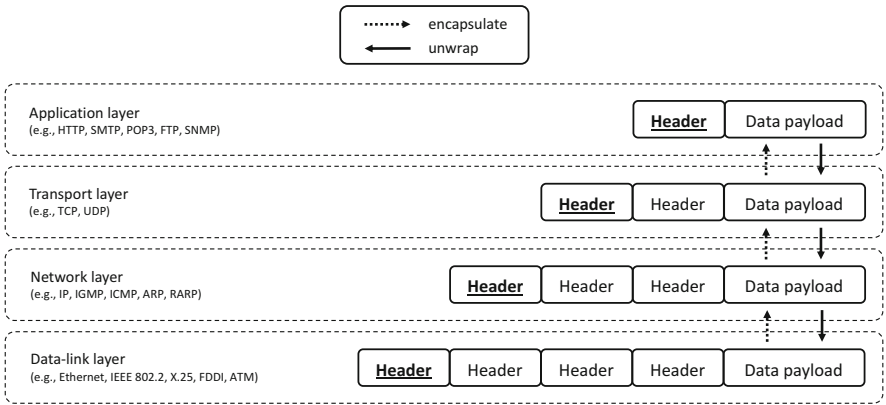


Fig. 4.8 IP Packet Encapsulation Process

Once the packets are prepared, the remainder of the packet switching process mainly consists primarily of routing them via the Internet and to their relevant destination. This process actually comprises two tasks: routing (i.e., choosing the path for a packet) and packet forwarding (i.e., the delivery of packets to another node). Specially configured systems, which can be either routers or switches, usually handle both tasks. A router acts as a gateway between two or more networks by connecting them physically and handling the packet exchange between them (Grance et al. 2008), which makes routers particularly important for the Internet's multi-network architecture. Switches serve a similar function, but instead of connecting different networks with one another, they interconnect nodes within the same network. For clarity's sake, the following explanations do not differentiate between routers and switches.

Router

On a network, a router is a device that determines the best path for forwarding a data packet toward its destination. A router is connected to at least two networks and is located at the gateway where one network meets another (Grance et al. 2008).

When receiving a packet, routers first perform basic error check routines. This includes, for instance, validating that a packet's checksum matches its content, which could indicate that its header or payload were modified in transit (Information Sciences Institute 1981). Routers also check whether a packet's time to live (TTL) limit has been exceeded. TTL limits a packet's lifespan by either defining a maximum number of transfers between routers, or a timespan to prevent a packet from circulating indefinitely. If the router cannot fix the packet, it is discarded and an error message sent to its sender. Depending on the type of error and the packet's history, the sender's TCP implementation may decide to resend the discarded packet or terminate the entire transmission.

If the packet is without errors, the router then determines its next routing target, also called a hop, on its way to the destination address specified in its header. This routing decision is made with the help of routing protocols that apply different algorithms to identify the best path to take, to facilitate establishing a network's structure, and to gather information about neighboring routers from the surrounding environment. Routing protocols also specify the handling of packets (e.g., error checking) and the forwarding process.

There are two major routing protocol archetypes, namely *distance vector routing* and *link state routing*. Distance vector routing often applies the Bellman–Ford and Ford–Fulkerson algorithms to calculate a graph comprising all of the available routes within or between networks. Each node determines the travelling cost to each immediate neighbor (Ford and Fulkerson 1956; Bellman 1958), because, in the routing context, these costs usually refer to metrics like the throughput, delay, transit fees, and reliability (Baumann et al. 2007). This information is gathered in respect of

each node and used to build a distance table, which allows for identifying the best path (i.e., the least costly) in support of a routing decision. The Interior Gateway Routing Protocol (IGRP) and Routing Information Protocol (RIP) are specific examples of distance vector routing protocols.

Link-state routing also tries to find the path with the lowest costs, but with a more local perspective. This means that a router based on distance vector routing cooperates with the other nodes, allowing each node to find the cheapest path to every other node linked directly to it. These data are then shared by the nodes and processed by using a standard shortest paths algorithm, such as Dijkstra's algorithm (Dijkstra 1959). Contrary to distance vector routing, which looks for the cheapest path between nodes A and C, link-state routing determines the least-cost path in a multiple of distinct paths (e.g., A to B and B to C). Such routing calculations are more effective, but also more time-consuming. The Open Shortest Path First (OSPF) and the Intermediate System to Intermediate System (IS-IS) routing protocols are examples of protocols using link-state routing.

Owing to the large number of packets a router has to handle, it is not practical to actually determine the best route by using complex algorithms for each incoming packet. Routers therefore use routing tables that store hop targets, which routing algorithms predetermined in order to identify the best next hop within the network or between different networks. To determine where a specific packet needs to be forwarded, a router simply has to find a suitable hop target within its routing table that directs the packet closer to its destined node. The routing table entry involved in the process does not necessarily need to be a precise match with the packet's destination IP address. A router may also use an approximate algorithm, like *Longest Prefix Match*, which identifies the closest destination based on subnetwork masks (i.e., longest match with the destination IP address's prefix) (Comer 2015). Once a suitable match is found, the packet is forwarded using the routing table information. This forwarding target is either another router that takes the packet closer to its destination, or the actual destination (if it is in reach of the router), which concludes the routing process.

If no appropriate forwarding target can be identified for a packet, it is sent on a default route with the expectation that it will find a router that can determine an appropriate route. This default route usually takes the packet up the ISP hierarchy (see section 4.2). The higher the network tier, the larger the maintained routing tables' size. Tier 1 ISPs often have the most extensive routing tables, which therefore have the best chance of determining where a packet should be sent. If an appropriate routing target is still not found, the packet will be discarded at some point (e.g., when reaching its TTL limit), which also concludes the routing process.

4.4 Content Delivery Networks

The reliable and fast delivery of content via the Internet is a challenging task due to the unpredictability of users' demand, which causes high server loads that exceed the available resources, and the large geographical distances between the content providers (i.e., entities that supply digital media, like texts, videos, and pictures, via the Internet) and the consumers across the globe. With the global Internet traffic continuously growing, load balancing approaches have become highly important for content providers trying to improve their customer experience (Robinson 2017). As the name implies, load balancing is an approach distributing the overall computational load burden (e.g., users requesting a website) evenly over multiple devices, ensuring that, at any given point in time, a single hardware or software component is not strained beyond its technical limitations. However, load balancing is not a function implemented at the Internet's core architecture level, as it was only designed with essential, but robust, network functionalities in mind (Saltzer et al. 1984). The implementation of more complex operations, like dealing with high load peaks caused by incoming resource requests, is the responsibility of nodes connected to the Internet.

An intuitive load balancing implementation is to build a large enough data center to host multiple redundant content servers with an appropriate load balancing mechanism to handle regular traffic and exceptional demand peaks. However, centralizing all computing resources in a single geographical location creates a single point of failure and does little to improve performance problems caused by network issues, large physical distances between a provider and customers, and power outages. Owing to these drawbacks, content providers settled for a different solution: content delivery networks (CDNs) (Pathan et al. 2008). Over the past decade, CDNs have become a vital part of the Internet architecture and by 2022 CDNs should be handling up to 72 percent of the global Internet traffic (Cisco 2018).

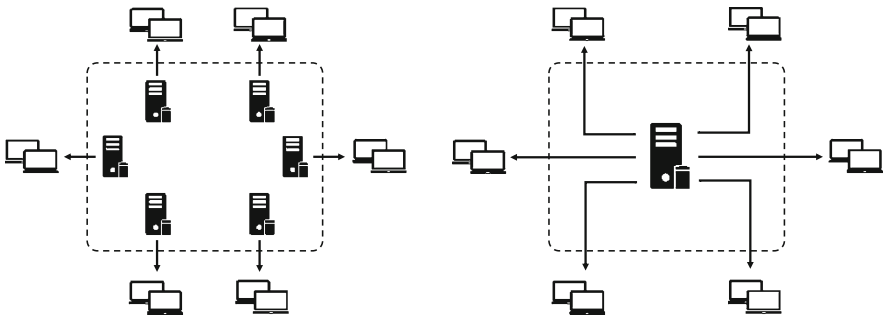


Fig. 4.9 (Left) Single Server Distribution (right) CDN Distribution Scheme

A CDN is a collection of network devices controlled by a common management infrastructure aimed at delivering content (e.g., websites, video streams, audio streams) more effectively via the Internet to content consumers (IETF 2003). The basic principle behind a CDN is to deliver content not from a central server, but to move it to network locations closer to content consumers (Fig. 4.9). By reducing the distance between the content and the consumer, CDNs decrease data transmission latencies, the risk of connection interruptions, and increase the transmission speed, which ultimately improves the user experience. This is done by replicating the content on multiple content delivery servers, also called surrogate servers, in different geographical locations (Cooper and Tomlinson 2001). Clients' content requests are automatically routed to the surrogate servers, a process often invisible to the requesting clients.

Content Delivery Network (CDN)

Content Delivery Network (CDN) is a type of content network in which the content network elements are arranged for more effective delivery of content to clients. A CDN usually comprises a request-routing system, surrogates, a distribution system, and an accounting system (IETF 2003).

A content provider (e.g., Netflix) either operates a CDN directly or a dedicated service provider acting on behalf of the content provider delivers content to its consumers. The latter is more common, as operating an own CDN with multiple points of presence in locations geographically far apart (i.e., close to the content consumers) involves high investment and maintenance costs (e.g., data centers, network infrastructure, labor costs). Moreover, most content providers' core business is the production of content, as they often have little expertise in the management of network infrastructures. Specialized network service providers, such as Amazon or Cloudflare, which sell CDN as a service to various content providers, therefore often run CDNs.

As depicted in Fig. 4.10, a CDN's architecture is basically comprised of four components, (1) a *content-delivery infrastructure*, (2) a *request-routing infrastructure*, (3) a *distribution infrastructure*, and (4) an *accounting infrastructure* (Hofmann and Leland 2005; Bartolini et al. 2003). The *content delivery infrastructure* contains the physical surrogate servers and the corresponding maintenance overheads. When a client requests a single content item from a CDN (e.g., a user request for a certain website), the request is directed to the best suited surrogate server with a copy of the item. As depicted in Fig. 4.10, a CDN could, for instance, operate one surrogate server per continent, each only handling requests from users located on the that continent.

The task of steering or directing a content request from a client to the right surrogate server is handled by the *request-routing infrastructure*. The selection of the "best suited" surrogate server usually means that the server promising the specific client the shortest delivery time with appropriate integrity and consistency

will deliver the item. The shortest delivery time is therefore not necessarily only dependent on static information, like a surrogate server’s geographic location and network connectivity. Consequently, the request-routing infrastructure often incorporates dynamic information, like the current network conditions and the surrogate servers’ computational load, into its decision on where to best direct user requests.

The *distribution infrastructure* handles activities concerning the content’s distribution within the CDN. All content delivered through a CDN is first published as a master copy on an *origin server*. A content item’s master copy is then replicated on one or more surrogate servers, which deliver copies of the content to a specific set of users. The distribution of the master copy to surrogate servers can either be implemented through pre-positioning (i.e., prior to user requests) or through on-demand fetching (i.e., in response to a user request to a surrogate server). If a content item needs to be modified or deleted, these changes are first applied to the master copy on the relevant origin server and afterwards replicated on the surrogate servers.

The final component is the *accounting infrastructure*, which is responsible for measuring and recording the networks’ content distribution and delivery activities. Keeping a record of these activities is very important, especially when the CDN delivers third-party content, since this information is used to calculate a client’s (i.e., the content provider’s) service fees.

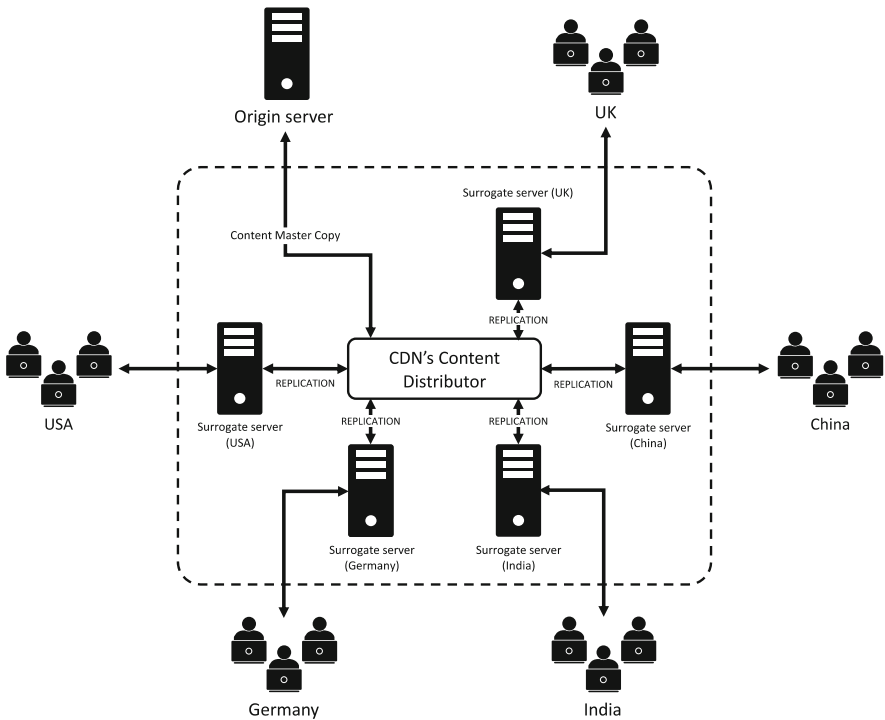


Fig. 4.10 Exemplary Structure of a CDN Distribution Infrastructure

In the CDN infrastructures discussed above, a content provider and content consumer are regarded as two different entities. However, this might not always be the case. For instance, a global retail organization can decide to build its own CDN to deliver content to its different subsidiaries across the globe. This type of network is called a **private CDN**, because its surrogate servers only serve content within the provider's (virtual) private network, which is restricted to certain business divisions, user groups, or persons. Depending on the organization's size, such a private CDN can range from just a pair of distributed caching servers to structures capable of serving thousands of content consumers (Savitz 2012). Large public CDNs sometimes also use private CDNs as part of their content distribution infrastructure to deliver content to their public surrogate servers (Google 2012). Private CDNs are most suited for organizations with specific location requirements (i.e., the placement of surrogate servers), distribution requirements (e.g. high bandwidth or low latency), or security requirements, like in healthcare (Sunyaev et al. 2010; Sunyaev and Pflug 2012; Dehling and Sunyaev 2014).

A **federated CDN**, which is based on an infrastructure operated by multiple content or service providers, is another CDN variation (Cisco 2011). Essentially, the participating providers pool their pre-existing resources in a single delivery network that enables its members to deliver their content to each node connected to the federated network. This means, for example, that when two national CDN operators A, in Australia, and B, in Belgium, agree to cooperate via a federated CDN infrastructure, provider A's content is delivered to content consumers in Belgium via B's national network, and vice versa. Content providers' interconnection can be implemented either bilaterally or via an exchange hub (Cisco 2011). In the former, each participating provider connects directly with every other provider in the network. When implementing an exchange hub, a central networking node orchestrates the content replication and delivery on behalf of all the providers. Regardless of the distribution approach, federated CDNs' main advantage is that smaller content or service providers can compete with larger CDNs, like Akamai or Limelight Networks, by reducing the operational complexity and costs for the individual providers and giving them more control over their content's distribution.

A **peer-to-peer (P2P) CDN** is a final example of an alternative CDN type. P2P describes a distributed application architecture in which equally privileged nodes (i.e., the peers) are directly interconnected with one another in order to share resources (e.g., services or data) (Dougherty et al. 2002). Unlike in the client-server architecture, in P2P networks there is no differentiation between resource providers and consumers, as each node fulfills both roles. The emergence of protocols, such as BitTorrent, used for sharing files without the need for a central server infrastructure has made P2P CDNs familiar to a wider audience of Internet users. In a P2P CDN, the network's clients (i.e., peers), which both provide and consume the content, either partly or completely replace surrogate servers. A P2P CDN creates a densely connected network with directly cross-linked Internet nodes to access the same content items. The CDN coordinates its clients to send chunks of an item to one another, instead of everyone sending individual requests to the same origin server. Consequently, the more content-consuming nodes use the network, the more effective P2P CDNs become, especially if the network is based on protocols, such as

BitTorrent, that require its nodes to share content while accessing it. Clients' involvement in the content delivery process can either be used to unburden existing surrogate servers or replace them completely. While this approach limits the content provider's control over the delivery process, it greatly reduces the CDN's setup and operational costs (Li 2008; Stutzbach et al. 2005).

In summary, each CDN type has its own distinct strengths and suitable application scenarios. However, all CDN approaches have enabling providers of popular content source, of content with fluctuating request patterns, and of a geographically distributed audience by means of a reliable and fast service for their content consumers in common.

4.5 Emerging Internet Network Architecture

The ongoing digitalization of today's society means that the demands placed on the Internet are growing as well. The Internet is therefore subject to continuous development and renewal. In the following, emergent architectural concepts will be presented that reflect the Internet network architecture's renewal process.

4.5.1 *Software-Defined Networking*

Software-defined networking (SDN) is an architectural approach for centralizing and simplifying computer networks' design and management by decoupling packet-forwarding activities (i.e., the data plane) from the decision process of how and where to send packets (i.e., the control plane) (Open Networking Foundation 2012). A similar functional distinction was made in section 4.3.4 when describing that network devices (routers or switches) first determine packets' optimal paths and only then forward them to adjacent nodes. In addition to routing decisions, the control plane is also responsible for network activities' monitoring and the enactment of the rules that the network operator sets, such as the traffic prioritization from or to certain nodes, or the security policies limiting access to parts of the network, or blocking specific packets due to their content. The different quality-of-service levels that the network operator sells to its customers require network devices to prioritize one customer's traffic over another in order to provide the promised network performance - a practical example of the abovementioned rules. The control plane is therefore of particular importance for network operators, since it defines the network's behavior and affects its overall performance.

A network's control plane is usually distributed across all the employed network devices. This means that once a device is configured, it autonomously gathers information about its surrounding network infrastructure by communicating with neighboring devices and subsequently makes routing decisions based on its preconfigured routing protocols. On the one hand, this distributed control approach makes the network more resilient due to its independence from a centralized control system (i.e., a potential single point of failure). On the other hand, distributing

control also entails the configuration being distributed. Network-wide configuration changes require manual reprogramming of each individual network device, which adds to the complexity of adapting a network's behavior to changing (application) requirements. Given that large computer networks (e.g., ISP networks) comprise hundreds if not thousands of networking devices by different manufacturers with different management interfaces, updating a network's configuration can be difficult and expensive. SDN's goal is to overcome this challenge by consolidating the control plane in one software-defined control system that instructs the network devices on how to forward packets and which can be centrally administered (Benzekki et al. 2017).

Software Defined Networking

Software Defined Networking (SDN) is an emerging network architecture in which network control is decoupled from the forwarding devices and is directly programmable (Open Networking Foundation 2012).

As depicted in Fig. 4.11, SDN defines three separate layers that interact with one another through standardized application programming interfaces (APIs). While the individual network devices located at the data layer still handle the forwarding of packets, all the control functions are moved from the individual hardware components to a single software controller (i.e., the SDN controller) that runs on a central system. Consequently, the SDN controller acts as a control hub enabling applications from the layer above to dynamically change how the individual network devices on the data layer below handle the network traffic. Two types of APIs establish communication between the application, control, and data layers: southbound and northbound APIs.

The southbound APIs connect the control and infrastructure layer. In the downward direction, the control layer sends configuration and routing information to the physical network devices (e.g., switches and routers). The network devices transmit operational data required for monitoring the network traffic up to the control layer. One of the API's main purposes is to control the routing of packets across network nodes. To ensure this control, the OpenFlow protocol, introduced in 2011, was the first standard used in southbound APIs and is currently still the most common (Little 2012, 2013). However, several alternative approaches, proprietary and open source, have been developed to create hardware abstraction layers, such as Cisco Systems' Open Network Environment and the Nicira Network Virtualization Platform.

The northbound APIs enable communication between the SDN controller and the software applications above that require network services to fulfill their intended function (Subramanian and Voruganti 2016). Such applications may utilize northbound APIs to customize the virtualized network infrastructure to their needs, and to access the network services that the control layer provides (Duan and Toy 2016). In contrast to southbound APIs, there is currently no standard protocol for northbound APIs. SDN often implements northbound APIs by using RESTful service interfaces based on the REST architectural style (see chapter 6).

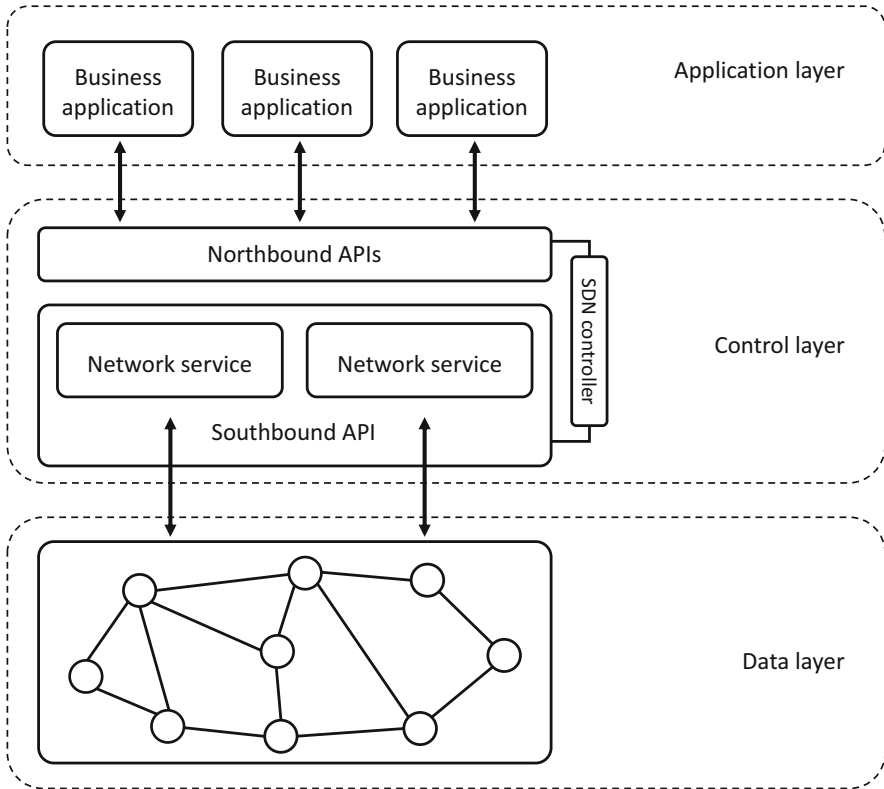


Fig. 4.11 Software Defined Network Architecture (adapted from Open Networking Foundation (2012))

One of the major advantages of SDN's separation of layers by means of standardized APIs is that the resulting loose coupling between the three layers enables virtualization and the dynamic allocation of network and service functions to an arbitrary underlying physical infrastructure. Consequently, SDN promises numerous benefits, including the physical network infrastructure's increased scalability (up and down), as well as the ability to monitor and adapt network resources quickly and precisely to exactly match the application and data needs. Another practical example of SDN benefits is the monitoring of network security. With the help of the centralized control layer, SDN operators can detect network anomalies caused by malicious behavior from inside or outside the network (e.g., unauthorized access to secured network nodes) much faster than in traditional, decentralized architectures. Once an anomaly has been detected, the affected network components, or whole sections, can be immediately reprogrammed to mitigate further damage (Giotis et al. 2014; Braga et al. 2010). Nonetheless, as mentioned previously, network controls' centralization also comes at the cost of introducing a single point of failure into the network, which may decrease its reliability (Benzekki et al. 2017).

4.5.2 *Network Functions Virtualization*

The idea of increasing network infrastructures' flexibility and agility through separation is a common theme in emerging network architecture approaches. This is true of SDN (see section 4.5.1), which separates control over a network and the network traffic's actual forwarding process. However, SDN alone does not affect the coupling of the network functions (e.g., intrusion prevention, load balancing, deep packet inspection) and purpose-built networking hardware, such as intrusion-detection devices, routers, and hardware firewalls. Non-virtualized network functions are usually implemented by combining specialized, proprietary software and hardware (ETSI 2013), which makes deploying new functions, which new services or applications require, in an existing network difficult. Owing to such networking devices' specificity, they often either need a fundamental reconfiguration, or even replacement, in order to execute new functions, which ultimately creates high investment and operational costs and makes it more difficult for network operators to provide dynamic networking services.

Network function virtualization (NFV) is an architectural principle that separates network functions from the hardware on which they run by using virtual hardware abstraction (ETSI 2018). Similar to the already common approach of virtualizing server hardware, NFV replicates the physical networking devices' functionality by using a software layer that can be used on any computing hardware. The European Telecommunications Standards Institute (ETSI), which is also responsible for the further development of the architectural approach, specified NFV in 2012 (Chiosi et al. 2012). As depicted in Fig. 4.12, the virtualized network functions (VNFs) implement network functions as software components that run on an NFV infrastructure. An NFV infrastructure comprises the physical hardware devices with computational, data storage and networking capabilities, which are assumed to be general purpose servers and storage devices (i.e., commercial off-the-shelf hardware). An additional virtualization layer, or hypervisor, decouples these capabilities from the hardware devices in order to provide a dynamic infrastructure that allows for evolving and scaling the VNFs and the hardware resources independently from one another. To control both the VNFs and the virtual infrastructure, NFV implements a management and orchestration system, which handles, for instance, the virtual and physical resources' coordination, the NFV infrastructure's monitoring, and the VNFs' deployment in the virtual infrastructure.

Network Function Virtualization

Network Functions Virtualization (NFV) describes the architectural principle of separating network functions (i.e., functional blocks within a network infrastructure that have well-defined external interfaces and well-defined functional behavior) from the hardware on which they run by using virtual hardware abstraction (ETSI 2018).

As mentioned at the outset, NFV and SDN are related approaches, as they both decouple logical functionality and physical devices, which is why the two

approaches can be combined to create a potentially greater value for the network operator. However, NFV can be implemented without SDN using the established server virtualization techniques used in many data centers. Vice versa, SDN does not necessarily require NFV approaches to separate the control and the data-forwarding planes. Nonetheless, NFV can facilitate SDN by delivering the infrastructure that allows the SDN to be implemented as a networking function.

NFV and SDN emphasize the utilizing off-the-shelf hardware and providing dynamic networking services (Chiosi et al. 2012). It is therefore not surprising that the benefits associated with the NFV approach are also closely aligned with SDN. NFV does not require purpose-built network hardware, since virtual networking devices can perform their tasks on basically any general-purpose computing device, such as commercially available workstations or servers. This often translates into large cost savings, because specialized hardware solutions are often produced in smaller quantities, which in turn leads to high prices per unit. However, network operators can easily repurpose existing computing resources for use in an NFV infrastructure, therefore also reducing the risks of the costly over- or under-provisioning of networking resources. The virtualization of networking devices can be added or subtracted when required, which means NFV also increases a network’s agility and its ability to adapt to shifting business requirements. Network operators can, for instance, tentatively test new configurations or services to incrementally meet their customers’ or their organization’s current needs without having to invest in new hardware components.

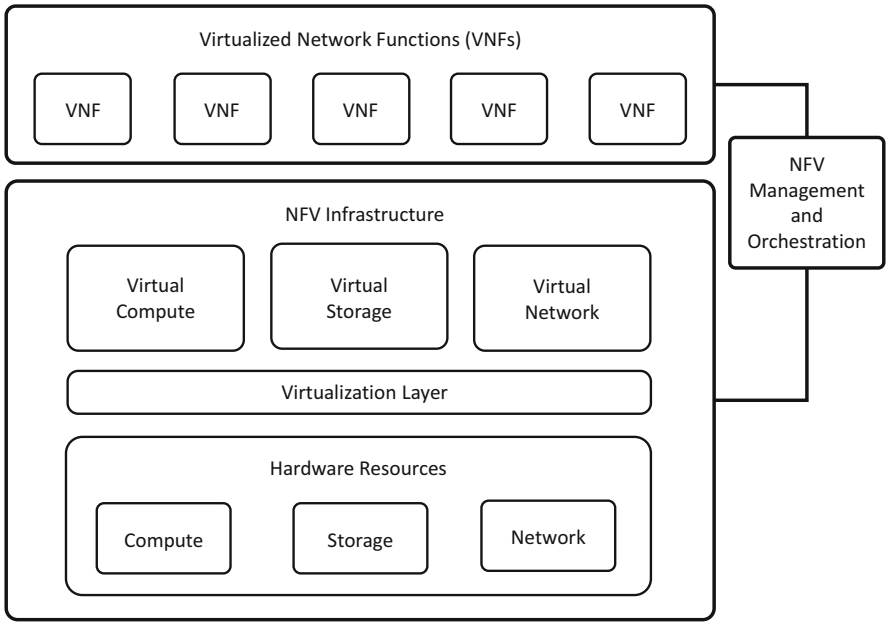


Fig. 4.12 High-level Architecture for Network Function Virtualization (adapted from ETSI (2013))

4.5.3 *Overlay Networks*

Overlay networks, also referred to as SDN overlays, are virtual networks of nodes, and the logical connections between them that run on an existing underlying network infrastructure (Bagad and Dhotre 2009). Overlay networks, which are somewhat similar to SDN, create an additional software layer that abstracts an underlying network's functionality. This software layer can then be customized to provide specialized services which are not available on the underlying network, or would require substantial reconfiguration of the network design (Malatras 2015). Overlay networks are therefore particularly useful when the underlying network cannot be easily customized to enable individual service offerings, which certainly applies to the Internet. It is not surprising that many well-known Internet services can be classified as overlay networks running on top of the public Internet. These services include, for instance, virtual private networks (VPN), peer-to-peer file sharing, and voice over IP (e.g., Skype). In the past, the Internet itself was an overlay network. As described at the beginning of the chapter, the Internet started as a research network interconnecting computer nodes via preexisting public telephone networks; in other words, the Internet overlaid the physical telephone infrastructure (Huang and Wu 2018). Currently it is the other way around, with telephone services turning into overlay networks on top of the Internet by switching to digital telephony solutions based on voice over IP (VoIP) technology.

Overlay Network

An overlay network is a virtual network of nodes and logical links built on top of an existing network in order to implement a network service not available in the existing network (Bagad and Dhotre 2009).

From a technical perspective, an overlay network uses software to create an abstract network layer overlying a physical network. This abstraction layer allows for defining overlay nodes on top of existing network nodes and the virtual connections between them. These overlay nodes need not correspond to the physical links between the underlying network nodes (see Fig. 4.13). The two or more involved nodes using the same software application layer that implements the extended network functionalities (e.g., encryption) create a virtual connection between themselves using a common overlay communication protocols, also known as tunnel protocols (Babay et al. 2017). These tunnel protocols basically encapsulate traffic inside IP packets, decoupling this traffic from the standardized TCP/IP processing logic, which in turn creates a virtual tunnel running through the network.

Software-defined virtual connections provide high customizability by allowing the implementing of purpose-built protocols that provide functionalities that the Internet does not support natively (without additional components), while still benefiting from its well established and scalable infrastructure. Without overlay networks, developers of innovative services might be forced to either create new

purpose-build networks or extend the current Internet infrastructure to natively (without additional components) support the functionalities required for their innovative applications (Babay et al. 2017). In comparison to these expensive and time-consuming endeavors, creating an overlay network is often a more efficient approach to adapting a network’s infrastructure.

However, there are also drawbacks that need to be considered when deciding to deploy or utilize an overlay network (Galán-Jiménez and Gazo-Cervero 2011; Sitaraman et al. 2014). In particular, the overlay abstraction layers, often stacked in multiples, add more protocols on top of the underlying network’s existing protocol stack (e.g., TCP/IP), thereby increasing the performance overhead costs and the communication complexity. Consequently, data exchange via an overlay network might require more network bandwidth due to the additional steps increasing the packet overhead costs and processing power when (de-) encapsulating the packets. Furthermore, the additional network complexity as a result of the multi-layer structure makes it more difficult to track down the root causes of performance or availability issues, which might be located at any of the virtual or physical networking layers (Galán-Jiménez and Gazo-Cervero 2011).

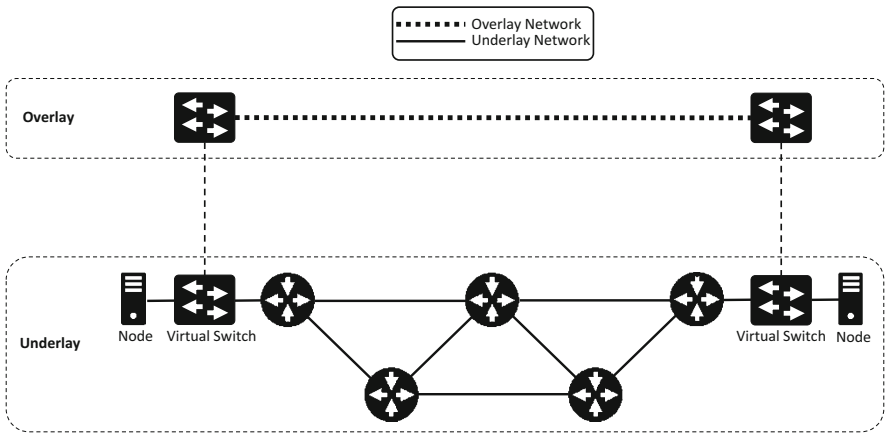


Fig. 4.13 Simplified Architecture of an Overlay Network

4.5.4 Information-Centric Networking

Today’s Internet architecture, which is based on TCP/IP, can be described as location-centric. Knowledge, in the form of an IP-address or domain name, is required about information’s and services’ location before such information and services can be accessed. It is not difficult to imagine that it would be far more convenient for Internet users to retrieve the desired information directly, instead of having to first search for the right website hosting the information. Further, this tight

bond between the data/services and the location makes maintaining discoverability more difficult when the location is changed, for instance, to decrease the physical distance to the users, increase the retrieval speed, and decrease the network traffic. However, this would require a paradigm shift away from TCP/IP and toward approaches that allow information and service discovery regardless of the distribution channel.

Such approaches are summarized under an architectural concept called information-centric networking (ICN). In ICN, *which* data are being communicated becomes more important than *who* is communicating them (Yaqub et al. 2016). Consequently, ICN probably envisions the most fundamental redesign of the existing Internet architecture compared to the other approaches described in this subchapter. In ICN, data become independent from their location, intended application, form of storage, and means of transportation, which enables innovative ways of data discovery and caching (Kutscher et al. 2016).

ICN can be implemented on different layers of the TCP/IP protocol stack. On the application and transport layers, ICN implementations may offer resource-naming overlays, location-independent content storage, and native content caching enabled by globally unique resource identifiers. On the network and data-link layers, ICN could act as a packet-level redesign of the current IP-based addressing of nodes, packet routing, and forwarding principles. One example of the latter is the content-centric networking (CCN) architecture (Jacobson et al. 2009), whose basic idea is that a user can broadcast an information need in the form of an *interest packet* specifying the sought after content's unique name. Depending on the specific CCN implementation, the content name may follow a hierarchical URI-based naming scheme. As depicted in Fig. 4.14, the interest packet is routed to a suitable content source, or cache, that can deliver the specified content interest (e.g., guided by a longest prefix match approach based on the URI). Each time, the interest passes a router, a record is created and stored in the router's pending interest table (PIT). These PIT records allow for tracing the packet's network path. Once a match is found between the proclaimed interest and the content object, the content object is enveloped in one or more *content packets* and routed back to the content consumer. The packets then take the reverse of the interest packet's path by following the previously created PIT records. With this approach, routing to and retrieval of content objects are not dependent on the object being held at a fixed location. This allows for a native implementation of content object caches at every CCN node, which drastically increases interest packets' chance of encountering a cached replica of the sought-after content object before reaching the content's origin node.

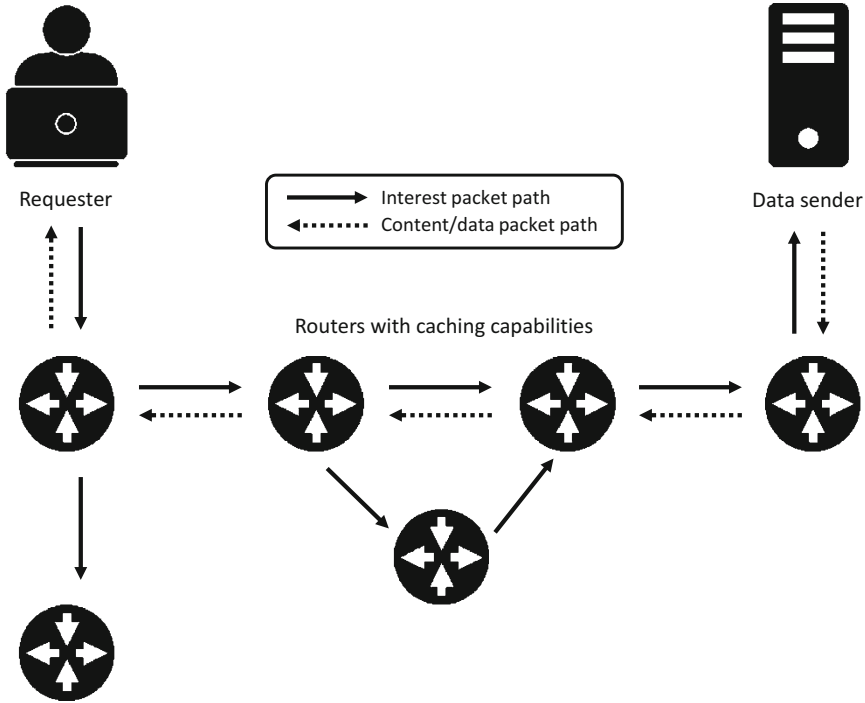


Fig. 4.14 Content-Centric Networking (CCN) architecture (adapted from Yaqub et al. (2016))

Another example of an ICN approach is the Publish-Subscribe paradigm, which consists of three basic elements: (1) information publishers, (2) information subscribers, and (3) a network of rendezvous points (RP) where the information exchange between the publisher and the subscriber takes place. The Publish-Subscribe Internet Routing Paradigm (PSIRP) was one of the early models, which envisioned a public subscription architecture (Tarkoma et al. 2009). In this paradigm, information items containing a data set and metadata are the architecture's core component, and are labelled and organized within scopes (Tarkoma et al. 2009). Scopes may denote a physical structure, like a corporate network, or represent a logical equivalent of the topology concept. An information item's location can therefore be specified by means of a rendezvous identifier (RId) and scope identifier (SId). The RId is the information item's unique identifier within a scope, whereas the SId denotes the scope in which an information item belongs (Fotiou et al. 2010). Scopes can also be nested within one another, which allows for creating flexible structures and increases the control over information dissemination, because scopes are responsible for an information item's policy and boundary enforcement (Tarkoma et al. 2009). To publish an information item, the publisher needs to know the SId of the scope in which it wants to publish its information. The information item is subsequently forwarded to the specific rendezvous point, which is hosted on a node in the network responsible for managing the scope

(Tarkoma et al. 2009). To obtain the information, the subscriber issues a subscription message consisting of the SId and RId, which are then mapped to the required information item's forwarding identifier (FId). The rendezvous function is responsible for finding a suitable data delivery path in the network and linking it with the FId. The intermediate routers use the FId to select the interface for forwarding the information. This label-based forwarding releases the router from maintaining the forwarding states and increases the network's scalability. Consequently, PSIRP can be implemented without relying on the IP protocol (Tarkoma et al. 2009; Fotiou et al. 2010).

ICN is an abstract idea for redesigning information retrieval with many potential implementation approaches, as described above. Despite these implementation approaches' differences, their common goal is to improve the performance and user experience of accessing information via the Internet and, at the same time, decreasing the Internet infrastructure's load (ICNRG 2018). Providing access to content and services by means of their names instead of their location, creates new and innovative caching approaches. Once information can be easily retrieved without having to know where it is stored, it becomes easier to implement extensive content caching and ICN-like infrastructures to shorten the paths between content providers and content consumers, which, in turn, increases the network performance and decreases the network traffic. Despite these benefits, ICN is not likely to replace location-centric network technologies in the near future, especially not the Internet whose success is rooted in agreed upon technological standards at its core architectural level. However, to obtain the previously mentioned benefits, ICN could be implemented as an overlay on top of the existing Internet architecture, which will then become more relevant for future content delivery and could even replace existing approaches like CDN (Kutscher et al. 2016).

Summary

With its roots in the 1950s as a U.S. military initiative, today's Internet architecture has come a long way. Some of the important milestones along this way were: the construction of the ARPANET (1969), the development of fundamental communication protocols like TCP/IP (1974) and HTTP (1989), the creation of the World Wide Web (1989), and the opening of the Internet to commercial service providers (1995). Since then, the Internet has become one of the most influential inventions in the history of mankind. Within 60 years it went from being a vague idea to an infrastructure used by more than four billion people around the globe and continues to grow and evolve through novel ideas, services, and architectures.

From an architectural perspective, the Internet can be described as a massive network of computer networks interconnect via the global Internet backbone. Commercial Internet Service Providers (ISPs) manage each subnetwork's operation and the interconnections between these networks. Depending on the managed network size, ISPs are classified into three tiers: tier 1 ISP (large-sized networks, national

scope, peer with other tier 1 ISP), tier 2 ISP (medium-sized networks, regional scope, peer with tier 2 ISP and purchase transit from tier 1 ISP), and tier 3 ISPs (small-sized networks, local scope, purchase transit from tier 1 and tier 2 ISPs). To establish communication between two endpoints via the Internet, up to three ISP levels must work together.

Non-profit organizations largely carry out the Internet's standardization and administration. These efforts include the standardization and further development of the Internet infrastructure (e.g., Internet Engineering Task Force), the administration and registration of Internet Protocol (IP) addresses for the different Internet regions (e.g., American Registry for Internet Numbers), the management of the domain name system (i.e., Internet Corporation for Assigned Names and Numbers), and the development and standardization of Web technologies (e.g., World Wide Web Consortium).

The Internet protocol suite, which specifies, for instance, how data are transmitted, packaged, addressed, routed, and received, is one of the cornerstones of the Internet architecture. This suite consists of four layers: the application layer, transport layer, network layer, and data-link layer. Each layer defines protocols, like the Transmission Control Protocol (TCP, transport layer) and Internet Protocol (IP, network layer), which, combined, provide end-to-end data communication via heterogeneous physical networks. This communication is based on a method called packet switching, which splits data transmissions into smaller data packets that are individually routed through the Internet to their recipient. To find the right recipient, each computer connected to the Internet is assigned a unique IP address, which consist of a 32-bit (IPv4) or 128-bit (IPv6) number. Devices called routers forward packets through the Internet based on their IP address. Each router sends a packet to a neighboring router closer to the packet's recipient until it reaches a router directly connected to a computer with a matching destination IP address. Since IP addresses are difficult to remember, they can be assigned text-based domain names. The Domain Name System (DNS) handles the process of translating the domain name into the associated IP address, which is called the DNS lookup.

When providers distribute content (e.g., websites, videos, or pictures) to their customers, they currently often employ Content Delivery Networks (CDNs). A CDN usually consists of a collection of surrogate servers that holds a copy of the content to be distributed and is strategically placed to minimize the physical distance between the content consumers and the surrogate server, which decreases the data transmission latencies, the risk of connection interruptions, and the transmission speed, thus ultimately improving the user experience.

Concluding this chapter, four emerging Internet network architectures were presented: software-defined networking (SDN), network function virtualization (NFV), overlay networks, and information centric networking (ICN). SDN centralizes control over a network by separating network packets' forwarding process (the data plane), routing process (the control plane), and applications utilizing network functions. This separation enables the network control to be directly programmed and the underlying infrastructure to be abstracted for applications and network services like server virtualization, cloud computing, and mobile networks.

Northbound and southbound APIs interconnect the separated planes. NFV replicates physical networking devices' (e.g., switches, load balancers, or routers) functionality by means of software components. Since these virtualized devices can run on any general-purpose hardware, NFV reduces costs by making specialized and expensive devices obsolete. NFV also makes network infrastructures more flexible, as it reduces the risks of networking resources' costly over- or under-provisioning. Overlay networks allow for operating discrete virtualized network layers on top of a physical network, thereby enabling new services or functions which would otherwise have required the network design's reconfiguration. This is of particular interest when the underlying network cannot be easily customized, which certainly applies to the Internet. Finally, ICN architectures shift the perspective from communication between two machines (location-centric) to content or service offerings identified by their names (content-centric). This paradigm change detaches information from its storage location, distribution system implementation, and means of transportation, thereby, simplifying information discovery and retrieval.

Questions

1. How can a computer network be classified?
2. What are the differences between the Internet and the WWW?
3. Is the Internet a private or public network? Substantiate your answer.
4. How do tier 1 and tier 2 ISPs differ?
5. How does IP-routing work?
6. How are domain names structured?
7. What are the individual steps of the DNS look-up process?
8. What are a content delivery network's main benefits?
9. What are the functions of the three SDN layers?
10. How does an overlay network work?

References

- Babay A, Danilov C, Lane J, Miskin-Amir M, Obenshain D, Schultz J, Stanton J, Tantillo T, Amir Y (2017) Structured overlay networks for a new generation of internet services. Paper presented at the international conference on distributed computing systems (ICDCS), Atlanta, GA, 5–8 June 2017
- Bagad VS, Dhotre IA (2009) Computer networks – II. Technical Publications, Pune, Maharashtra
- Baker FJ (2009) Core protocols in the internet protocol suite. <https://tools.ietf.org/id/draft-baker-ietf-core-04.html>. Accessed 19 Sept 2019
- Bartolini N, Casalicchio E, Tucci S (2003) A walk through content delivery networks. Paper presented at the international workshop on modeling, analysis, and simulation of computer and telecommunication systems, Orlando, FL, 12–15 Oct 2003
- Baumann R, Heimlicher S, Strasser M, Weibel A (2007) A survey on routing metrics. <http://rainer.baumann.info/public/tik262.pdf>. Accessed 19 Sept 2019

- Bellman R (1958) On a routing problem. *Q Appl Math* 16(1):87–90
- Benzekki K, El Fergougui A, Elaloui AE (2017) Software-defined networking (SDN): a survey. *Secur Commun Netw* 9(18):5803–5833
- Berners-Lee T, Bray T, Connolly D, Cotton P, Fielding R, Jeckle M, Lilley C, Mendelsohn N, Orchard D, Walsh N, Williams S (2004) Architecture of the world wide web, vol 1 W3C. <https://www.w3.org/TR/webarch/>. Accessed 19 Sept 2019
- Braga R, Mota E, Passito A (2010) Lightweight DDoS flooding attack detection using NOX/OpenFlow. Paper presented at the IEEE local computer network conference, Denver, CO, 10–14 Oct 2010
- Cerf V, Kahn R (1974) A protocol for packet network interconnection. *IEEE Trans Commun* 22(5):637–648
- Cerf V, Sunshine C (1974) Specification of internet transmission control program. <https://tools.ietf.org/html/rfc675>. Accessed 19 Sept 2019
- Chandramouli R, Rose S (2006) Challenges in securing the domain name system. *IEEE Comput Soc* 4(1):84–87
- Chapin L (1992) Charter of the internet architecture board (IAB). <https://tools.ietf.org/html/rfc1358>. Accessed 19 Sept 2019
- Chiosi M, Clarke D, Willis P, Reid A, Feger J, Bugenhagen M, Khan W, Fargano M, Cui C, Deng H, Benitez J, Michel U, Damker H, Ogaki K, Matsuzaki T, Fukui M, Shimano K, Delisle D, Loudier Q, Koliass C, Guardini I, Demaria E, Minerva R, Manzalini A, López D, Salguero FJR, Ruhl F, Sen P (2012) Network functions virtualisation: an introduction, benefits, enablers, challenges & call for action. Paper presented at the SDN and OpenFlow world congress, Darmstadt, 22–24 Oct 2012
- Cisco (2011) Content delivery network (CDN) federations: how SPs can win the battle for content-hungry consumers. https://www.cisco.com/c/dam/en/us/products/collateral/video/videoscape-distribution-suite-service-broker/cdn_vds_sb_white_paper.pdf. Accessed 19 Sept 2019
- Cisco (2018) Cisco visual networking index: forecast and trends, 2017–2022. <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>. Accessed 19 Sept 2019
- Comer D (2015) Computer networks and internets, 6th edn. Pearson Education, London
- Cooper I, Tomlinson G (2001) Internet web replication and caching taxonomy. <https://tools.ietf.org/html/rfc3040>. Accessed 19 Sept 2019
- Deering S, Hinden R (1995) Internet protocol, version 6 (IPv6): specification. <https://tools.ietf.org/html/rfc1883>. Accessed 16 Sept 2019
- Deering S, Hinden R (1998) Internet protocol, version 6 (IPv6): specification. <https://tools.ietf.org/html/rfc2460>. Accessed 19 Sept 2019
- Deering S, Hinden R (2017) Internet protocol, version 6 (IPv6) specification. <https://tools.ietf.org/html/rfc8200>. Accessed 16 Sept 2019
- Dehling T, Sunyaev A (2014) Information security and privacy of patient-centered health it services: what needs to be done? Paper presented at the 47th Hawaii international conference on system sciences, Waikoloa, HI, 6–9 Jan 2014
- Dijkstra EW (1959) A note on two problems in connexion with graphs. *Numer Math* 1(1):269–271
- Dougherty D, Truelove K, Shirky C, Dornfest R, Gonze L (2002) 2001 P2P networking overview: the emergent P2P platform of presence, identity, and edge resources. O'Reilly, Sebastopol, CA
- Duan Q, Toy M (2016) Virtualized software-defined networks and services. Artech House, London
- ETSI (2013) Network function virtualisation (NFV); architectural framework. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf. Accessed 19 Sept 2019
- ETSI (2018) Network functions virtualisation (NFV); terminology for main concepts in NFV. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.04.01_60/gs_NFV003v010401p.pdf. Accessed 19 Sept 2019
- Falk A (2009) Definition of an internet research task force (IRTF) document stream. <https://tools.ietf.org/html/rfc5743>. Accessed 19 Sept 2019

- Fall KR, Stevens RW (2011) TCP/IP illustrated, vol 1. Addison-Wesley, Ann Arbor, MI
- Ford LR, Fulkerson DR (1956) Maximal flow through a network. *Can J Math* 8(1):399–404
- Fotiou N, Nikander P, Trossen D, Polyzos GC (2010) Developing information networking further: from PSIRP to PURSUIT. Paper presented at the international conference on broadband communications, networks and systems, Athens, 25–27 Oct 2010
- Frazer KD, Merit Network Inc., National Science Foundation (1996) NSFNET: a partnership for high-speed networking: final report, 1987–1995. Merit Network, Ann Arbor, MI
- Galán-Jiménez J, Gazo-Cervero A (2011) Overview and challenges of overlay networks: a survey. *Int J Comput Sci Eng Surv* 2(1):19–37
- Giotis K, Argyropoulos C, Androulidakis G, Kalogeras D, Maglaris V (2014) Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Comput Netw* 62(7):122–136
- Gokhale AA (2005) Introduction to telecommunications. Thomson/Delmar Learning, Clifton Park, NY
- Google (2012) Inter-datacenter WAN with centralized TE using SDN and OpenFlow. <https://www.opennetworking.org/wp-content/uploads/2013/02/cs-googlestdn.pdf>. Accessed 19 Sept 2019
- Grance T, Hash J, Peck S, Smith J, Korow-Diks K (2008) Security guide for interconnecting information technology systems. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-47.pdf>
- Hinden R, Deering S (2006) IP version 6 addressing architecture. <https://tools.ietf.org/html/rfc4291>. Accessed 19 Sept 2019
- Hofmann M, Leland RB (2005) Content networking: architecture, protocols, and practice. The Morgan Kaufmann series in networking. Morgan Kaufmann, San Francisco, CA
- Hornig C (1984) A standard for the transmission of IP datagrams over ethernet networks. <https://tools.ietf.org/html/rfc894>. Accessed 19 Sept 2019
- Huang D, Wu H (2018) Mobile cloud computing: foundations and service models. Elsevier, Cambridge, MA
- Hunt C (2002) TCP/IP network administration, 3rd edn. O'Reilly Media, Sebastopol, CA
- ICNRG (2018) Information-centric networking research group. <https://irtf.org/icnrg>. Accessed 1 Dec 2018
- IETF (1989a) Requirements for internet hosts – application and support. <https://tools.ietf.org/html/rfc1123>. Accessed 16 Sept 2019
- IETF (1989b) Requirements for internet hosts – communication layers. <https://tools.ietf.org/html/rfc1122>. Accessed 16 Sept 2019
- IETF (2003) A model for content internetworking (CDI). <https://tools.ietf.org/html/rfc3466>. Accessed 16 Sept 2019
- Information Sciences Institute (1981) Transmission control protocol. <https://tools.ietf.org/html/rfc793>. Accessed 16 Sept 2019
- Jacobson V, Smetters D, Thornton J, Plass M, Briggs N, Braynard R (2009) Networking named content. Paper presented at the 5th international conference on emerging networking experiments and technologies, Rome, 1–4 Dec 2009
- Jordana J (2002) Governing telecommunications and the new information society in Europe. Edward Elgar, Cheltenham
- Kozierok CM (2005) The TCP/IP guide: a comprehensive, illustrated internet protocols reference. No Starch Press, San Francisco, CA
- Kutscher D, Eum S, Pentikousis K, Psaras I, Corujo D, Saucez D, Schmidt T, Waehlich M (2016) Information-centric networking (ICN) research challenges. <https://www.rfc-editor.org/pdf/rfc/rfc7927.txt.pdf>. Accessed 16 Sept 2019
- Li J (2008) On peer-to-peer (P2P) content delivery. *Peer Peer Netw Appl* 1(1):45–63
- Little RG (2012) Software-defined networking is not OpenFlow, companies proclaim. <https://searchnetworking.techtarget.com/news/2240158633/Software-defined-networking-is-not-OpenFlow-companies-proclaim>. Accessed 4 Sept 2019

- Little RG (2013) InCNTRE's OpenFlow SDN testing lab works toward certified SDN product. <https://searchnetworking.techtarget.com/news/2240186631/InCNTREs-OpenFlow-SDN-testing-lab-works-toward-certified-SDN-product>. Accessed 4 Sept 2019
- Malatras A (2015) State-of-the-art survey on P2P overlay networks in pervasive computing environments. *J Netw Comput Appl* 55(1):1–23
- Mansfield KC, Antonakos JL (2009) Computer networking for LANS to WANS: hardware, software and security. Cengage Learning, Boston, MA
- Mills DL, Braun H (1987) The NSFNET backbone network. *ACM SIGCOMM Comput Commun Rev* 17(5):191–196
- Mockapetris P (1983a) Domain names – concepts and facilities. <https://tools.ietf.org/html/rfc882>. Accessed 16 Sept 2019
- Mockapetris P (1983b) Domain names – implementation and specification. <https://tools.ietf.org/html/rfc883>. Accessed 16 Sept 2019
- NIST (2013) Security and privacy controls for federal information systems and organizations. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf>. Accessed 16 Sept 2019
- Norton WB (2011) The internet peering playbook: connecting to the core of the internet, 2nd edn. DrPeering Press, Palo Alto, CA
- Open Networking Foundation (2012) Software-defined networking: the new norm for networks. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Accessed 16 Sept 2019
- Pathan M, Buyya R, Vakali A (2008) Content delivery networks: state of the art, insights, and imperatives. In: Buyya R, Pathan M, Vakali A (eds) Content delivery networks. Lecture Notes electrical engineering, vol 9. Springer, Berlin, pp 3–32
- Piliouras TC, Terplan K (1998) Network design: management and technical perspectives. CRC Press, London
- Robinson D (2017) Content delivery networks: fundamentals, design, and evolution, 1st edn. Wiley, Hoboken, NJ
- Saltzer JH, Reed DP, Clark DD (1984) End-to-end arguments in system design. *ACM Trans Comput Syst* 2(4):277–288
- Savitz E (2012) Netflix shifts traffic to its own CDN; Akamai, Limelight Shrs Hit. <https://www.forbes.com/sites/ericsavitz/2012/06/05/netflix-shifts-traffic-to-its-own-cdn-akamai-limelight-shrs-hit/>. Accessed 4 Sept 2019
- Sitaraman RK, Kasbekar M, Lichtenstein W, Jain M (2014) Overlay networks: an Akamai perspective. In: Pathan M, Sitaraman RK, Robinson D (eds) Advanced content delivery, streaming, and cloud services, 1st edn. Wiley, Hoboken, NJ, pp 305–328
- Stevenson A, Waite M (2011) Concise Oxford english dictionary, 12th edn. Oxford University Press, New York, NY
- Stewart B (2000) CSNET – computer science network. https://www.livinginternet.com/i/ii_csnet. Accessed 4 Sept 2019
- Stutzbach D, Zappala D, Rejaie R (2005) The scalability of swarming peer-to-peer content delivery. Paper presented at the international conference on research in networking, Waterloo, ON, 2–6 May 2005
- Subramanian S, Voruganti S (2016) Software-defined networking (SDN) with OpenStack. Packt Publishing, Birmingham
- Sunyaev A, Pflug J (2012) Risk evaluation and security analysis of the clinical area within the German electronic health information system. *Heal Technol* 2(2):123–135
- Sunyaev A, Leimeister JM, Krcmar H (2010) Open security issues in German healthcare telematics. Paper presented at the 3rd international conference on health informatics (HealthInf), Valencia, 20–23 Jan 2010
- Tarkoma S, Ain M, Visala K (2009) The publish/subscribe internet routing paradigm (PSIRP): designing the future internet architecture. In: Tselentis G, Domingue J, Galis A et al (eds) A European research perspective. IOS Press, Amsterdam, pp 102–111

W3C (2017) W3C mission. <https://www.w3.org/Consortium/mission>. Accessed 4 Sept 2019

Yaqub MA, Ahmed SH, Bouk SH, Kim D (2016) Information-centric networks (ICN). In: Ahmed SH, Bouk SH, Kim D (eds) Content-centric networks: an overview, applications and research challenges. Springer, Singapore, pp 19–33

Further Reading

Berners-Lee T, Bray T, Connolly D, Cotton P, Fielding R, Jeckle M, Lilley C, Mendelsohn N, Orchard D, Walsh N, Williams S (2004) Architecture of the world wide web, vol 1 W3C. <https://www.w3.org/TR/webarch/>. Accessed 19 Sept 2019

Comer D (2015) Computer networks and internets, 6th edn. Pearson Education, London

Hunt C (2002) TCP/IP network administration, 3rd edn. O'Reilly Media, Sebastopol, CA

Mansfield KC, Antonakos JL (2009) Computer networking for LANS to WANS: hardware, software and security. Cengage Learning, Boston, MA

Singh MP (2005) The practical handbook of internet computing. CRC Press, Boca Raton, FL

Tanenbaum AS, Van Steen M (2017) Distributed systems: principles and paradigms, 2nd edn. Prentice-Hall, Upper Saddle River, NJ

Chapter 5

Middleware



Abstract

In the context of IT applications and especially in large organizations, integration of existing information systems into new IT environments poses many challenges. One of the biggest issue in this regard is dealing with the systems' heterogeneity in terms of used programming languages, operating systems, or even data formats. In order to ensure communication between different information systems, developers must establish common interfaces. This chapter introduces middleware as a type of software which manages and facilitates interactions between applications across computing platforms. Besides a brief definition and overview of middleware, several of its characteristics are described. Furthermore, the differences between the three middleware categories (message-oriented, transaction-oriented and object-oriented middleware) are defined. In addition to these theoretical foundations, some practical implementations are presented.

Learning Objectives of this Chapter

The primary learning objective of this chapter is to flesh out the concept, history, and current use of middleware. The chapter's main goal is to point out the difference between transaction-oriented middleware, message-oriented middleware, and object-oriented middleware. The subchapters present the key characteristics of each of these categories and their commercial implementation. Students will also learn about transaction processing monitors and remote procedure calls. The latter will be explained step by step and contrasted with local procedure calls. Finally, the chapter provides information about the well-known CORBA standard.

Structure of this Chapter

The chapter's first section defines the overall concept of middleware and gives some brief historical background. This section is followed by a description of the best-known type of middleware: the remote procedure call. The third section forms the main part of this work and gives an overview of the different categories of middleware: message-oriented middleware, transaction-oriented middleware, and object-oriented middleware. Definitions, functionalities, commercial implementations, and well-known standards like CORBA are all included in the middleware categories' descriptions. The final section summarizes the whole chapter.

5.1 Introduction to Middleware

In today's digitally interconnected world, infrastructures are becoming more complex and diverse than ever before. At the same time, developers are facing increasing demands from an equally heterogeneous assortment of consumers around the globe for more powerful, reliable, and efficient applications. One way to facilitate the development process and create more flexible and reliable applications is to use sophisticated middleware. Middleware refers to application-neutral programs that mediate communication between applications in such a way that the complexity of these applications and their infrastructure is hidden from the user. Middleware can also be understood as a distribution platform (Ruh et al. 2002): It provides a solution to developers seeking to integrate a collection of servers and applications into a common service interface. From an application perspective, middleware is a service layer, which is used instead of an operating system interface. This is an appropriate course of action if the services offered are more powerful or the interface guarantees platform independence. For instance, in the context of mobile computing, middleware facilitates the development of platform-independent applications that are developed once but can be used across different mobile operating systems (e.g., Android and iOS) or even accessed via the Web.

At present, the term tends to be associated with a specific service area and three competing technologies. Two of these technologies are Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM). The most commonly used programming language, Java, contains a third, competing technology called Remote Method Invocation (RMI). These technologies will be discussed later in the chapter.

Middleware systems enable the distribution of applications to multiple computers in the network. The concept of 'Web services,' which will be introduced in Chapter 6, can be seen as a special type of middleware systems. Web services are an extension of the middleware concept, especially in terms of cross-organizational data exchange, which is of particular interest in business-to-business integration. The present chapter focuses on conventional types of middleware systems, which are explained in detail in the subchapters below.

Middleware

Middleware is a type of software used to manage and facilitate interactions between applications across computing platforms.

In other areas, the term middleware has a different meaning. For example, in the field of computer game development, middleware may refer to subsystems for areas like game physics. Such middleware is often produced and offered by third-party developers.

The term itself has been in use since the late 1960s (Naur and Randell 1986). The understanding of middleware as a solution to the problem of linking newer

applications to older legacy systems gained popularity in the 1980s. However, the answer to the question of what makes something middleware has changed over time. The definition depends on the usage context and the level of abstraction in the technology stack, which can range from the physical infrastructure and networking components to the application and user interfaces. Application developers consider everything below the application programming interface (API) as middleware. Networking experts see everything above IP as middleware. Those working on applications, tools, and mechanisms between these two extremes see it as located somewhere between TCP and the API, with some classifying middleware even further as application-specific upper middleware, generic middle middleware, or resource-specific lower middleware. On frequent occasions, the point is made that middleware often extends beyond the ‘network’ to the computer, storage capacity, and other network-related resources (Aiken et al. 2000).

Middleware is also an essential building block of distributed systems. As shown in Fig. 5.1, the entire distributed system is a set of applications geared towards the clients. In this example, the middleware goes unnoticed by the clients, but it is responsible for connecting them with the servers. The middleware is connected via an API to a set of applications. An API is a program part that a software system (middleware in this example) makes available to other programs in order to connect them in a simple way. APIs are software system programs that make particular functionalities or services visible to clients while abstracting the details of how these services are implemented. In this way, clients do not need to know about the technical specifications of the underlying technical platforms. For example, the middleware routes client requests to the appropriate servers.

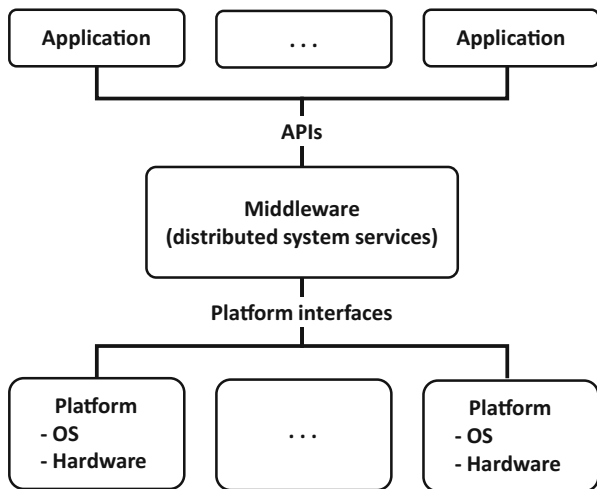


Fig. 5.1 Middleware in a distributed system (adapted from Bernstein (1996))

Middleware tends to be a highly developed communication component connecting client applications to servers. It can provide a wide range of services at different levels of transparency and functionality and is a type of software that uses defined interfaces or messages to facilitate the communication of requests between applications. In addition, middleware provides the runtime environment to manage the requests between applications (Ruh et al. 2002). It mediates between applications in such a way that the complexity of these applications and their infrastructure is concealed, and it offers services above the transport layer (i.e., TCP/IP) services but below the application environment (Aiken et al. 2000). Middleware can also be understood as a distribution platform (i.e., as a protocol or a protocol bundle on a layer higher than that of ordinary computer communication). While lower-level network services handle simple communication between computers, middleware supports communication between applications.

Behind the idea of middleware is the core problem of integrating not only proprietary applications' legacy applications but also their data, which are developed, distributed, and run on different hardware/software platforms and for different purposes. It is becoming increasingly necessary to develop holistic, integrated applications from existing functions and components. In the Internet age in particular, software concepts and technologies are needed for continuous adaptation and further development and to abstract from the details of the relevant implementation. The primary goal of middleware is to master distributed application systems' complexity by means of a uniform abstract intermediate layer. This basic idea goes back to the early stages of computer science and first appeared in print in the late 1960s. However, it was only from the 1980s onward, with the emergence of networking and distributed systems and the associated complexity of increasingly cross-company and cross-organizational applications, that middleware became an important and central research field and a developmental focus of (business) computer science. Since the 1990s, a number of commercial and non-commercial middleware concepts and products have been available. They form the basis for the development of complex distributed applications today.

In general, middleware has to offer a range of services and interface functions that allow distributed applications to interact and cooperate across network boundaries. Important aspects include the provision of local transparency, which means hiding the distribution, as well as independence from the specifics of the programming language, networks, communication protocols, operating systems, and hardware (see Fig. 5.2).

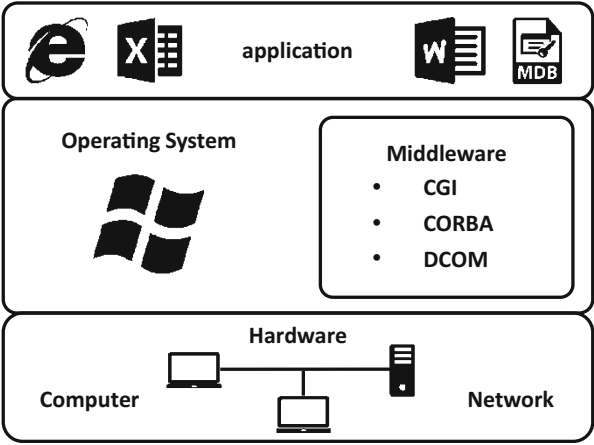


Fig. 5.2 Middleware in a layer model

Middleware works at a high level within the layer model. It transports complex data, connects function calls between the components, and establishes transaction security. Middleware software is available as standard from several companies, including IBM, Oracle, and Microsoft. On the application layer, it might be impossible for application A to communicate with application B. For various reasons, this communication could be rooted in different programming languages. Applications that want to use the middleware layer to communicate with each other can use these interfaces. The middleware software component will forward the corresponding calls via a network. Usually, standard protocols like IP and TCP are used.

5.2 Remote Procedure Call

Remote Procedure Call (RPC) is the most basic type of middleware (Alonso et al. 2004). It allows for functions to be called in other address spaces (frequently on another computer on a shared network). Usually, the called functions and the calling program are not executed on the same computer. There are many implementations of this technique, and they tend not to be compatible with each other.

Remote Procedure Call

A remote procedure call is the synchronous language-level transfer of control between programs in disjoint address spaces whose primary communication medium is a narrow channel (Nelson 1981).

The general idea of a procedure call fits the request–response pattern of the client–server model (see Chapter 2). The client makes a request to some external code, sleeps, and finds the result after control is returned to the procedure. Thus, procedure calls constitute a natural programming abstraction for the request–response message exchange pattern.

James E. White presented the first concepts of RPC in a technical report in 1976 (White 1976), and in 1994 Andrew Birrell and Bruce Nelson received the ACM Software System Award for developing RPC (ACM 2017). The most widely used variant is the Open Networking (OPC) RPC, which is often referred to as Sun RPC because it was originally developed by Sun Microsystems for its Network File System (NFS). For this RPC variant, there is also an implementation in Linux. Besides ONC RPC, the Distributed Computing Environment (DCE) RPC is also widely used. Microsoft derived Microsoft RPC (MSRPC) from the DCE RPC 1.1 reference implementation. Microsoft’s proprietary DCOM technology was later implemented on this basis. DCOM-related experiences contributed to the development of .NET Remoting.

In order to understand procedure calls, it is essential to understand the functioning and merits of RPC. These calls are usually bound to a process’s address space, which is allocated by the operating system. Hence, a procedure can only be called if it is linked to an address in this space. The idea behind calling a procedure is to make it accessible at a certain address in a process’s address space. Luckily, in high-level languages such as C/C++, Java, C#, etc., developers do not have to take care of all the technical details of the procedure call since the compiler automatically generates the machine-level instructions for the procedure call. ‘Normal’ procedure calls are limited to procedure calls in the same address space, like the calling code, so they can only make procedure calls within a process. To solve this problem, RPC offers a generalized notation of a procedure call to other processes – for example, a client call to a procedure in a remote process or even in a remote machine. Developers typically use the term to refer to remote procedure calls, regardless of whether the procedure resides in a different process on the same machine or in a process on another machine. However, in the context of Microsoft’s DCOM middleware, the notion of local procedure calls (LPCs) is well-known but misleadingly refers to calls of a remote procedure hosted in a different process on the same local machine. The reason for this distinction is that communication with processes on the same machine can be implemented in more efficient ways since no communication via the network is required (see Fig. 5.3). However, such a ‘local’ remote procedure call is still not the same as a (normal) procedure call within the same address space because it involves inter-process communication between different address spaces.

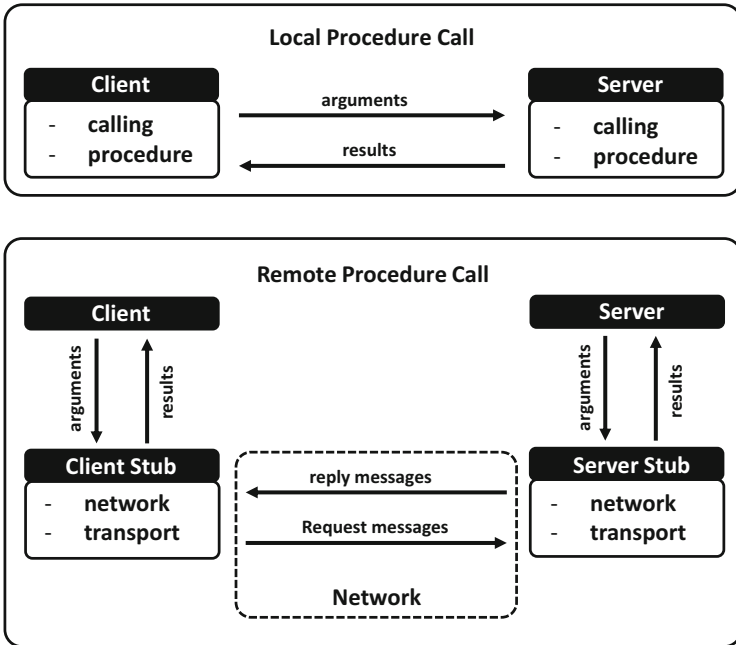


Fig. 5.3 Local procedure call vs. remote procedure call (adapted from Muppidi et al. (1996))

It can, however, be supported by the operating system kernel, for instance, by copying chunks of memory from the address space of one process to the address space of another process. This still imposes an overhead, but the overhead is much smaller than for a remote machine.

Besides local procedure calls, the use of RPC in the context of distributed systems follows a common approach (Alonso et al. 2004). To set up an RPC, developers have to use the interface definition language (IDL). IDL is used to set up communications between clients and servers in RPC. Developers use an IDL to specify the interface between a client and a server so that the RPC mechanism can create the code stubs needed to call functions across the network. Stubs are interfaces that simulate the existing code's behavior. The stubs are interfaces between the client and the server and make the called procedures work like local procedures on both sides. To create code stubs, any middleware using RPC or a similar concept provides an interface compiler, which typically produces client stubs, server stubs, code templates, and references (Alonso et al. 2004).

The stub may be created in the user's program and comprise remote calls of procedures prior to the program run (preprocessing). After it has been created, the stub resides in the operating system, a library, or a runtime system of the user's program (Czaja 2018). The client and server stubs are subprograms consisting of two modules tasked with: (1) transmitting and (2) receiving. Communication between the client and the server is done by stubs on both sides (see Fig. 5.4). It means the

stubs do not contain the procedures as such. In other words, the stub is a placeholder or proxy for the actual procedure implemented at the server.

The client stub transforms the procedure call into the required form to the server to reproduce it on the server side, execute the call to obtain the results, and, finally, reproduce these results on the client side. When the client calls a remote procedure, the call that is actually executed is a local call to the procedure provided by the stub. The stub then locates the server (i.e., it binds the call to a server), formats the data appropriately (which involves marshaling and serializing the data), communicates with the server, receives a response, and forwards that response as the return parameter of the procedure invoked by the client (Alonso et al. 2004). In this case, marshaling is the packing of procedure parameters into a message packet, which could be interpreted by the systems involved (Huang et al. 2010).

The server stub is the counterpart to the client stub. It implements the server side of the invocation and also receives invocations from the client side. The received invocation is formatted in the format for the procedure on the server side and forwarded to the client stub as a response from the server, which is generated as a result of the procedure on the server side.

Additionally, the RPC's IDL compiler generates the header files needed to do the compiling. Modern IDL compilers even go one step further: They can also generate templates with the basic code for the server, such as programs containing only procedure signatures but no implementation. The developer simply needs to add the code that implements the procedure at the server and code the client as needed (Alonso et al. 2004).

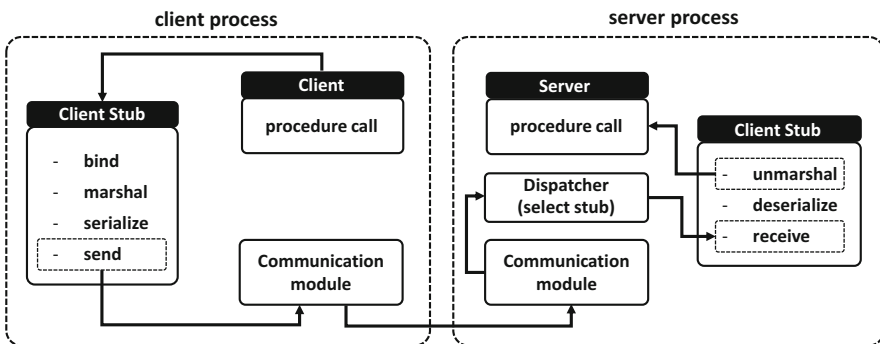


Fig. 5.4 Functions of remote procedure calls (adapted from Alonso et al. (2004))

In addition to the specification of message formats and sequence, the RPC is tasked with marshaling the arguments. In other words, procedures' arguments and return values must be encoded into a bit stream that can be packed into a message's payload. This process is referred to as marshaling because it resembles the assembly of individual wagons (the arguments) to a train (the message). Since the communication is based on a procedure call, the RPC should be able to encode any language

data type that can be used as an argument for a local procedure. Obviously, the middleware running on the client and the server side must use the same message format and encoding; there must be agreement about the protocol.

Implementation of the underlying communication is based on lower-level programming abstractions, such as sockets. This means that, before calling a remote procedure, a client must know the server's address before initiating a connection. Similarly, the server has to bind a socket to a specified address and port. This process is referred to as binding: Before making calls, the client has to be bound to the server. There are two different ways to do this: (1) static and (2) dynamic binding. The simplest way is to use static binding: The developer can start the client with the specific server address or even hardcode the server address (and port) into the client at compile time. The advantage is that a call can be made directly without any overhead or need to search for the server. However, it requires that the developers reconfigure or even recompile the client whenever the server is relocated. Load balancing and an automated failover mechanism that can compensate for the crash of a server are impossible in the case of static binding. By contrast, dynamic binding schemes locate the server at runtime before making a call. This requires an additional lookup service, which allows the server to register under some name and which can then be used by the client to find the address and port of the server. This allows developers to decouple the (symbolic) name from the (location-dependent) address, which is similar to using domain names instead of physical hardware addresses on the Web (see Chapter 4). Using dynamic binding schemes facilitates load balancing and the operation of a failover mechanism but requires an additional service and imposes a comparably small overhead for the lookup.

5.3 Middleware Categories

Literature provides several categorizations and descriptions of middleware (Alonso et al. 2004; Curry 2005; Bernstein 1996; Mahmoud 2005). These different categories result from the historical development of middleware and its ongoing improvement in commercial implementation. This chapter sorts middleware into three main categories. The first subchapter shows the functionalities and some commercial implementations of message-oriented middleware infrastructures. The next subchapter shows the aspects of transaction-oriented middleware and the well-known implementation of the transaction processing (TP) monitor. The last subchapter defines the characteristics of object-oriented middleware and the established CORBA architecture.

5.3.1 *Message-Oriented Middleware*

MOM refers to middleware based on synchronous or asynchronous communication, namely, the transmission of messages. A MOM system client can send messages to

and receive messages from other clients of the messaging system. The format for MOM messages is not fixed, but XML (see chapter 6) has become a popular format. Each client connects to one or more servers that act as intermediaries in the sending and receiving of messages.

Message-Oriented Middleware

Message-oriented middleware (MOM) is any middleware infrastructure that provides messaging capabilities. It provides a means to build distributed systems, where distributed processes communicate through messages exchanged via message queuing or message passing (Curry 2005; Bouchenak and de Palma 2009).

MOM uses a peer-to-peer relationship between individual clients; each peer can send messages to and receive messages from other client peers. MOM platforms facilitate the creation of flexible cohesive systems; a cohesive system is one that allows changes in one part of a system to occur without the need for changes in other parts of the system (Curry 2005). XML is widely used as the basic language for messages in MOM. Owing to the comparatively self-explanatory and (unlike binary format messages) easily human-readable format, it is relatively easy to use XML to enable communication between middleware systems if they use different languages, as long as the languages are XML-based. To enable communication, an XSLT processor (XML is discussed in depth in Chapter 6) can be used as an intermediary translator to translate messages from the source system's XML-based language to the target system's language by means of a transformation style sheet. SOAP (see Chapter 6) is often used as the protocol. The idea of MOM was not new but was often presented as a revolutionary technique that could be a game changer in the context of developing distributed systems. Some RPC implementations already offer a way to carry out asynchronous interactions, and some TP monitors (see chapter 5.3.2) contain message queues to handle message-based interactions (Alonso et al. 2004).

Message Passing vs. Message Queuing

Message passing refers to transient communication between two processes that are active at the same time. In other words, the receiver needs to be ready to receive a message when the sender sends the message. For example, asynchronous RPC is based on a (transient) passing of the request message. 'Asynchronous' indicates that the client does not wait for a response.

The message queuing model requires an additional intermediary component, which is called the message queue (essentially a mailbox independent of the receiver). The message queue stores messages and, thus, temporally decouples sender and receiver. The use of persistent storage for the buffering of messages makes it possible to extend a message's lifetime beyond the lifetime of the message queue process. Thus, messages can still be used even in the event of a message queue failure.

The differences between message passing and message queuing are illustrated in Fig. 5.5. The times when processes are active (i.e., when they are ready to send

and/or receive messages) are depicted on the time axis as a bold line. In the message passing example on the left, the rendezvous time – that is, the time during which both processes are active and can exchange messages – is visually indicated by the gray area. In the message queuing example on the right, there is no need for such a rendezvous time because of the intermediate message queue. This allows processes to communicate even though they are not active at the same time. A simple analogy of this is two basketball players passing a ball to each other without the ball touching the ground. However, the message queuing model would allow one player to place the ball on the ground and leave it there for another player to pick up (either immediately or later). Some forms of MOM offer publish/subscribe communication as an alternative form of message queuing and message passing. Publish/subscribe is a model of network programming in which a message sender does not explicitly specify the receiver's address (Uramoto and Maruyama 1999).

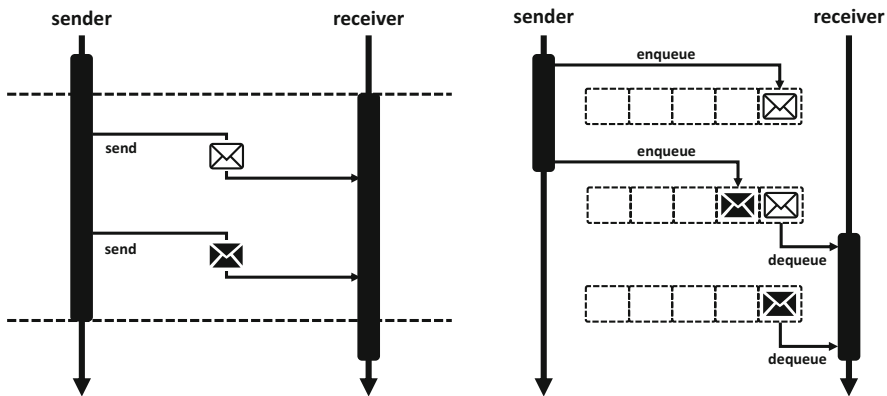


Fig. 5.5 Message passing vs. message queuing

Message Broker

Any MOM that handles messages to route them to the specific recipient uses a message broker as a building block (Clark et al. 2002). The MOM provides basic communication among the applications and features, including message persistence and guaranteed delivery. A message broker is an intermediary process that implements message queues. Fig. 5.6 shows the function of a message broker. Both the sender and the receiver of a message only need to directly exchange messages with the intermediary broker. This message broker can run on a central machine, there can be a system of federated message brokers, or it can even be outsourced to a cloud computing provider. The sender can issue an enqueue request, which means the broker stores a message in a given message queue (see Fig. 5.6). When the recipient is ready to receive the next message from a given queue, it sends a dequeue request. In turn, the message broker will return the message and delete it from the queue. Necessarily, the communication between sender and broker – and between receiver and broker – is implemented based on message passing.

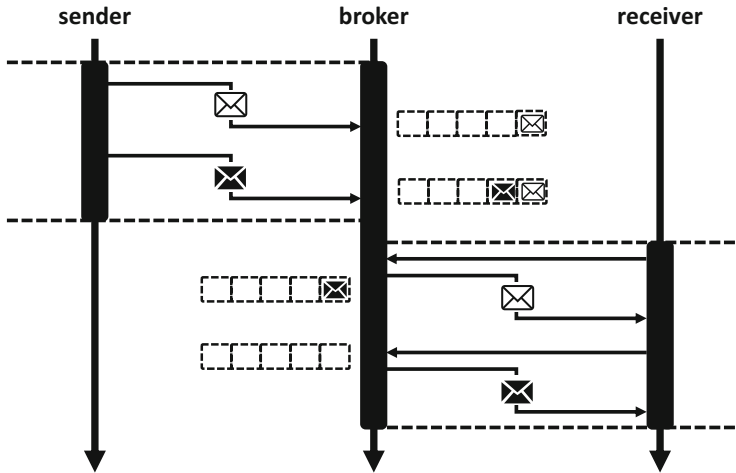


Fig. 5.6 Message queuing with a message broker

When distributed systems are designed, implementing a message broker offers a number of additional benefits. First, the client (or sender) does not need to be aware which system the server (or receiver) is running. It just sends messages to the message broker, which adds the message to a particular queue instead of sending it directly to the server (or receiver). So, in a message broker implementation environment, the sender focuses on the message's semantics rather than who the recipient is. Second, the receiver can change its location during the runtime without affecting the client. This means the location dependency between a client and a server is reduced. It can also be transparently changed to multicast or broadcast messages: The message broker can decide not to delete the message from the queue after the first dequeue operation and could, instead, decide that a certain group of recipients should receive it. This simplifies the scalability significantly because there could be multiple servers (labeled as receivers in Fig. 5.7) without the client even being aware of it.

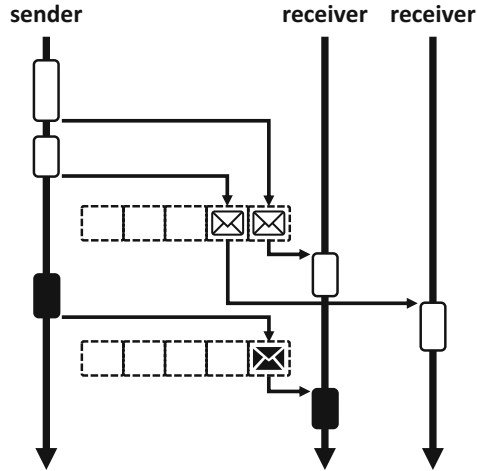


Fig. 5.7 Multiple receivers in a message broker scenario

Publish/Subscribe

As mentioned above in the first section, a MOM could offer publish/subscribe communication instead of message queuing or message passing. This special case of message queuing is called publish/subscribe message-oriented middleware (PSMOM) (Wang et al. 2011; Albano et al. 2014). A PSMOM's messages are events 'published' to recipients who have 'subscribed' to receive them (Albano et al. 2014). Publish/subscribe is a messaging pattern that categorizes published messages into classes without the subscribers' knowledge (Belokosztolszki et al. 2003). Similarly, subscribers express interest in one or more classes and only receive events that match these classes without having any knowledge of the senders (Eugster et al. 2003). PSMOMs can be topic-based, content-based, or data-based. In a topic-based system, the subscribers can receive events based on individual topics, which are described by keywords. In this interaction paradigm, the topic is the message queue, and each topic builds its own queue (Albano et al. 2014). The content-based system enhances the topic-based approach by introducing a subscription scheme based on the content (Cugola and Jacobsen 2002; Shen 2010). The data-based approach is an advanced version of the topic-based approach in which the data structures are shared among peers (senders and subscribers). Sharing the data structures enables the peers to receive only the changed data. If a new peer subscribes to these events, only the final status of the data will be sent instead of all the events regarding the requested topic (Albano et al. 2014).

Messaging Protocols

When MOM first became popular, a lack of standards limited the interoperability between different MOM implementations. Today, in order to reduce technological dependence on the implementation, language, and platform, MOM uses standardized APIs and protocols. Standards like Java Messaging Service (JMS), Advanced

Message Queuing Protocol (AMQP), and Data Distribution Service by the Object Management Group (OMG DDS) now enhance the interoperability of MOM implementations. JMS is part of the Java Platform Enterprise Edition (Java EE) and was defined by a Sun Microsystems specification. The specification of the service and the associated API has been standardized by the Java Community Process within the framework of JSR 914 (Hapner 2002). It is a messaging standard that allows applications based on Java EE to create, send, receive, and read messages from MOM (Curry 2005). Owing to JMS's widespread use, all of its functions have been incorporated into the JSR 914 framework or current MOM standard. This allows developers to continue using the JMS interface, while MOMs can communicate with each other by means of AMQP (OASIS 2019). OMG DDS was developed by the Object Management Group. It is a comprehensive standard that includes a standardized API and a set of standardized protocols. It is based on the publisher/subscriber concept, which supports deterministic resource management. The specification is divided into two areas: Data-Centric Publish/Subscribe (DCPS), which describes the basic concepts for data distribution, and Data Local Reconstruction Layer (DLRL), which provides an abstraction layer for applications based on DCPS (Object Management Group Inc. 2019).

Implementations of MOM

Some examples of commercial implementations of MOM are Message Queuing (MSMQ) (Microsoft 2016), Apache ActiveMQ (Apache Software Foundation 2019), Amazon Simple Queue Service (SQS) (Amazon Web Services 2019), IBM WebSphere MQ (IBM 2019), and Rabbit MQ (Pivotal Software 2019). MSMQ is an application protocol developed by Microsoft that provides message queues. On Windows, it is deployed by the Microsoft Message Queue Server (Microsoft 2016). Apache ActiveMQ is a free message broker that fully implements Java Message Service 1.1 (JMS). Apache ActiveMQ changes the connections of a network between existing applications by converting synchronous communication between applications to be integrated into asynchronous communication (Apache Software Foundation 2019). Amazon SQS was introduced by Amazon.com in late 2004. It supports the programmatic sending of messages via Web service applications to communicate over the Internet. SQS is intended to provide a highly scalable hosted message queue that resolves issues arising from the common producer-consumer problem or connectivity between producer and consumer (Amazon Web Services 2019). IBM WebSphere MQ is a platform-independent MOM. It was introduced in 1992 and is based on the principle of message queuing. By default, it offers OAM (object authority manager) and secure sockets layer (SSL) security for communication (IBM 2019). RabbitMQ is an open-source MOM that implements the Advanced Message Queuing Protocol (AMQP) and essentially falls under the control of VMWare. It has been extended with plug-in architecture to support the Streaming Text-Oriented Messaging Protocol (STOMP), and Message Queuing Telemetry Transport (MQTT). The RabbitMQ server is written in the Erlang programming language. The Open Telecom Platform (Pivotal Software 2019) serves as the foundation for clustering and failover mechanisms. The main differences

between these implementations of MOM are different programming languages, different levels of scalability, varying kinds of administration functions, and several backup functions.

5.3.2 *Transaction-Oriented Middleware*

Transaction-oriented middleware (TOM) supports synchronous and asynchronous communication among heterogeneous hosts and facilitates integration between servers and database management systems (Capra et al. 2001). TOM is often used with distributed database applications.

Transaction-Oriented Middleware

TOM is any middleware infrastructure that supports the execution of electronic transactions in a distributed setting (Alonso 2018).

The following subchapter explains the concept of TOM, including one of the oldest and best-known types of TOM: the TP monitor. The subsequent subchapter describes many of the TP monitor's functionalities and aspects. Historically, the TP monitor has grounded TOM, which runs on mainframes to provide functionalities that are not offered by the operating system (Alonso 2018).

Concept of Transaction-Oriented Middleware

TOM enables developers to define the services offered by server components. The client applications that they develop can request several of those services that are offered on the server component. The client and server components can be implemented on different hosts because the transaction can be transported via the network. The way in which the transactions are transported is transparent to both the client and the server components. In addition, the client component's requests can be either synchronous or asynchronous. TOM supports various activation policies. The service can be permanently active or active on demand, and if a service has been idle for too long, it can be deactivated (Capra et al. 2001).

TOM guarantees the atomicity property of a transaction as long as the participating systems implement the two-phase-commit protocol (see Fig. 5.8). In the first phase of the standardized two-phase commit, a coordinator (usually the process that initiates the committing) obtains approval or denial to commit the data changes of all the processes involved (also called the 'Prepare Phase'). Only then, if all participants agree, does the coordinator decide to 'Commit'; otherwise, there is a 'Rollback' of the decision. If the decision has been made, the coordinator informs the participants of the result in the second phase ('Commit Phase') of the protocol.

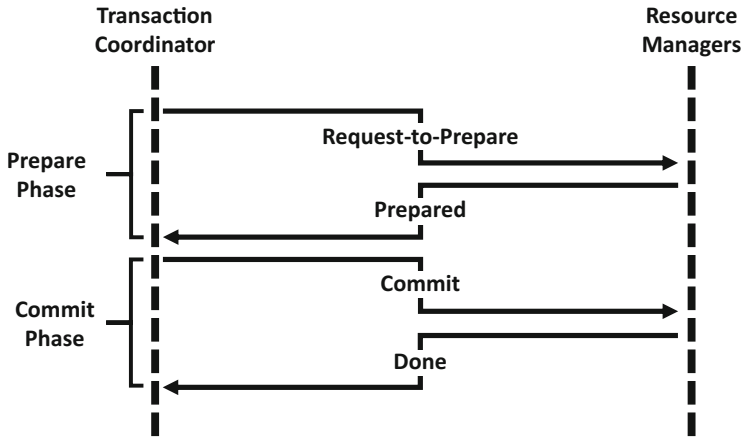


Fig. 5.8 Two-phase commit

According to this common result, either the entire transaction is reset or all partial transactions are brought to a successful conclusion by releasing the resources that have been blocked in the process (Bernstein et al. 1995). In addition to these functionalities, most TOM implementations offer a load balancing and replication of server components (Balasubramanian et al. 2004).

TOM implementations are still a key component of many enterprises' computing solutions and are mainly used in the context of high-performance processing, for example, in the context of financial services such as online trading. They are also used in application servers – for example, online shops or multitier systems – to support the execution of transactions like purchases (Alonso 2018).

ACID Transactions

To support transactions, TOM has to use existing transaction-processing concepts like ACID. ACID, which stands for atomicity, consistency, isolation, and durability, is a set of properties for transactions that are sometimes called classical transactions or flat transactions (Mahmoud 2005). They are a prerequisite for system reliability. This acronym was coined in 1983 by computer scientists Theo Härder and Andreas Reuter (Haerder and Reuter 1983).

In the context of TOM, a transaction is a series of database operations that are executed either in full or not at all. They are also known as all or nothing affairs (Mahmoud 2005). In practice, the individual database statements for the transactions are executed sequentially. These database statements are not validated and put into effect before they have been successfully completed. If the transaction cannot be completed during the process, the origin area is declared valid and a rollback is performed. This means that all previously executed statements are reversed if necessary, or the database's memory area, which is used for the changes in the meantime, is released, and the validity is left at the previous statement. The two-phase commit (see Fig. 5.8) and the DO-UNDO-REDO protocol form the

basis for achieving atomicity in many TOMs. The DO-UNDO-REDO protocol is used to roll back or roll forward transactions with the help of transaction log entries. Atomicity guarantees that each transaction is treated as a single ‘unit’ that either succeeds or fails in its entirety. Consequently, transactions should be designed as do, undo, and redo functions, where the do function produces a log record that can be used in the case of an error during the transaction to undo or redo this transaction (Gray 1981). Consistency means that a transaction leaves a consistent database state after completion if the database was consistent before. This implies that all integrity conditions defined in the database schema are checked before the transaction is completed (Cattell 2011). The property of isolation states that transactions should be designed in such a way that they prevent transactions that are executed in parallel from influencing each other. This is usually achieved by locking procedures that obstruct the required data for other transactions before data are accessed. Locking procedures limit concurrency and can lead to blocking. In many database systems, the isolation method that is used can be configured to achieve greater concurrency. The most common problem is the ‘lost update’ anomaly, in which two transactions read and update the same object at the same time. This can lead to data being overwritten by the second transaction, causing the data of the first transaction to be lost. The transactional isolation level defines the permitted type of influence (Mahmoud 2005). Property durability means there is a guarantee that the data will be permanently stored in the database after a transaction has been completed successfully. The permanent storage of the data must also be guaranteed after a system error. In particular, data must not be lost if the main memory fails. Durability is ensured by the DO-UNDO-REDO protocol, writing a transaction log (write-ahead log protocol), and enforcing the ‘force-at-commit’ rule. The combination of these protocols and rules allows the user to reproduce all missing write operations in the database after a system failure.

The ‘write-ahead log’ (WAL) protocol is a database technology procedure that helps to ensure transactions’ atomicity and durability. It states that modifications must be logged before they are actually written. It enables an ‘update-in-place,’ which means that the old version of a data record is overwritten by the new version at the same location (Haerder and Reuter 1983). According to the ‘force-at-commit’ rule, a transaction should not be committed until the after-images of all its update parts are in stable storage (in this case, in the log or the database) (Bernstein and Newcomer 2009).

Transaction Processing Monitor

The transaction processing (TP) monitor is one of the oldest and best-known types of TOM. Initially, TP monitors were developed as multithreaded servers to support large numbers of terminals from a single process. The TP monitor is also the most reliable, most stable, and best-tested technology in the enterprise application integration context (Alonso et al. 2004). TP monitors are not developed for general program-to-program communication integration but provide solutions for transaction-type applications that utilize a database (Khosrowpour 2002). IBM’s Customer Information and Control Systems (CICS) was one of the earliest TP

monitor implementations, and it was the first commercial product offering transaction-protected distributed computing (Gray and Reuter 1992). It was built at the end of the 1960s and is still in use today (e.g., in banking transactions) (Alonso et al. 2004).

For many decades, the TP monitor completely dominated the middleware market. It is one of the most successful forms of middleware and makes it possible for many operations in our everyday lives, such as purchasing a plane ticket by connecting several airline systems, to deliver good performance and be reliable. TP monitors can still be found in most application servers and Web services implementations today (Alonso et al. 2004). Besides CICS, there are many commercially offered implementations, including Microsoft Transaction Server (MTS) and Oracle Tuxedo, which was developed by AT&T in the 1980s. MTS is based on proven transaction processing methods. MTS comprises a simple programming model and execution environment for distributed component-based server applications (Limprecht 1997). Tuxedo (Transactions for Unix, Extended for Distributed Operations) is a transaction processing system, transaction-oriented middleware, or enterprise application server for a variety of systems and programming languages (Bernstein and Newcomer 2009).

The TP monitor's main function is to coordinate the flow of requests between terminals or other devices and application programs that can process these requests (Fig. 5.9). A request is a message that asks the system to execute a transaction. Transactions can be processed in parallel by several TP monitors or TP monitor processes (performance and scaling). The application that executes the transaction usually accesses resource managers, such as database and communications systems. A typical TP monitor instantiation includes functions for transaction management, queuing, routing, and messaging and supports the distribution of applications. Many TP monitors also provide locking, logging, and recovery services to enable application servers to implement ACID (see the section above) properties by themselves. Transaction management involves support for operations that start, commit, and abort a transaction. It also provides interfaces to resource managers (e.g., database systems) that are accessed by a transaction so that a resource manager can tell the transaction manager when it is accessed by a transaction, and the transaction manager can tell resource managers when the transaction has committed or aborted. A transaction manager implements the two-phase commit protocol. This protocol ensures that all or none of the resource managers commits are executed. The queuing function of the TP monitor enables the persistent storage of data to move between transactions. A distributed queue manager can be invoked remotely and may provide the system management operations required to forward elements from one queue to another (Bernstein 1996). Distributed queuing means sending messages from one queue manager to another. A queue manager handles all incoming messages. The routing function ensures that messages from the client side are sent to the servers by using TP monitors (Chorafas 1998), and the messages are coordinated by TP monitors and handled via the queuing function mentioned above (Ray 2009).

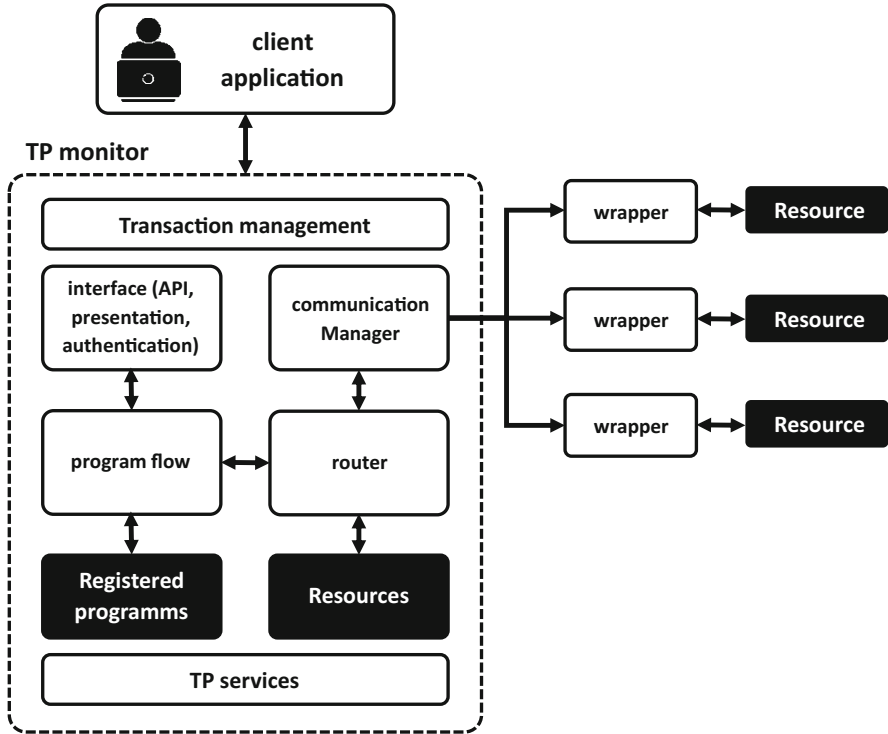


Fig. 5.9 Components of a TP monitor (adapted from Alonso et al. (2004))

By integrating different services, the TP monitor offers a simplified and uniform API. One way to simplify the API is to maintain the context and thereby prevent certain parameters from being passed on. For instance, a TP monitor generally maintains the context of the actual request, transaction, and user. Most of the application functions do not have to specify this information. Rather, the TP monitor completes the necessary parameters when it converts an application call to a call on the underlying middleware. For instance, an application may request that a message be queued, and the TP monitor will add the current transaction ID and the user ID as parameters. A TP monitor can customize the user interface. For instance, it can have a stylized interface that displays errors and menus and makes it possible to log in. A TP monitor has application development tools, including a data dictionary for sharing record definitions between form managers, application programs, and databases. A TP monitor also has system management tools that display the state of a transaction or determine which components are currently unavailable. System management can be deployed in its own framework, which combines the TP monitor with the platform and database system so that all resources can be managed in the same way from the same device. Most clients buy a complete TP system from a vendor, including a TP monitor, database system, and platform (Alonso et al. 2004).

TP-Lite and TP-Heavy Monitors

In the context of TP monitors, making a distinction between TP-lite and TP-heavy monitors is a common way to describe different implementations (Gray 1993). When the TP monitor implementation is integrated in a database system, it is called a TP-lite monitor. The stored procedure is invoked and executed according to the ACID properties. In contrast to the TP-lite monitor, the TP-heavy monitor implementation supports the client-server architecture and allows the user to initiate very complex transactions (Gray and Reuter 1992). They act as a standalone product for the development of distributed applications that require high performance (Alonso 2018). TP-heavy monitors offer flexibility by using standardized transaction protocols.

The two kinds of monitors both have advantages and disadvantages. For example, if the data are stored in multiple databases, a TP-lite solution is not appropriate because it does not support complex transactions (Umar 2004). If the data synchronization interval is periodic, a TP-lite monitor is better than a TP-heavy solution. In general, small applications are more likely to use a TP-lite monitor (Umar 2004). One commercial implementation of TP-heavy and TP-lite monitors can be found in Oracle RDBMS. Oracle Tuxedo Oracle RDBMS is a database management system software owned by Oracle and offers a TP-lite monitor functionality, while Oracle Tuxedo offers a TP-heavy monitor (Oracle 2019a).

Transactional RPC

TP monitor implementations can be considered an extension of the RPC concept. They handle remote procedure calls in a transaction with the inherent ACID properties. In particular, the ACID property of atomicity means that either all invoked remote procedure calls are processed or none of them are. Thus, TP monitors implement an abstraction of RPC, called transactional RPC (TRPC). In TRPC, a group of procedure calls is provided with the transactional brackets beginning of transaction (BOT) and end of transaction (EOT) and treated as a unit. This is the task of the transaction management module, which controls the interactions between clients and servers and ensures their atomicity by implementing the two-phase-commit protocol (Alonso et al. 2004).

5.3.3 Object-Oriented Middleware

Object-oriented middleware (OOM) is based on the RPC mechanism. This subsection provides a general definition of OOM and introduces CORBA, which is the best-known OOM standard. In the literature, the OOM concept is not clearly defined and is sometimes also referred to as request broker, object broker, object request broker, or distributed object middleware (Gokhale 2009; Mahmoud 2005; Alonso et al. 2004). These terms are sometimes used interchangeably but can also describe distinct or related OOM concepts.

Object-Oriented Middleware

OOM is defined as a middleware infrastructure that offers object-oriented principles for the development of distributed systems (Capra et al. 2001).

Concept of Object-Oriented Middleware

OOM enables communication between objects within a distributed system (e.g., the Internet) and is independent of both the operating system and the programming language. In short, OOM infrastructures support interoperability among objects and provide location transparency through RPC (Birrell and Nelson 1984). The idea of OOM is to make the principles of object-oriented programming, such as object identification through references and inheritance, available for the development of distributed systems. OOM supports distributed object requests from clients, which means that a client can initiate the operation of a server object that may reside on another host. To this end, the client defines an object reference in relation to the server object. Stubs, which are generated from interface definitions (written in IDL), marshal the operation parameters required to execute a function on the server object and handle the results. Every OOM uses the IDL. In addition to standard RPC, an OOM IDL supports object types as parameters, failure handling, and inheritance. OOM enables IDL compilers to generate client and server stubs, which implement the session and presentation layer. OOM provides very powerful component models that integrate the functionalities of TOM, MOM, and RPC (Capra et al. 2001).

Interface Definition Language

The IDL is a declarative formal language containing a language syntax to describe a software component's interfaces. It can be used to describe objects and the methods applicable to them, together with the possible parameters and data types, without using the properties of a particular programming language. The interface description language serves purely to describe the interface but not to formulate algorithms. Starting with the IDL, a special compiler can convert the definitions into a specific programming language and computer architecture, which is called language binding (Wirsing and Nivat 1996).

Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA), developed by the OMG (Object Management Group Inc. 2012), is a well-established specification for OOM. At its core is an Object Request Broker (ORB), which defines cross-platform protocols and services. The first version, CORBA 1.0, was published in August 1991. The CORBA specification is not bound to a specific platform; rather, software manufacturers or communities are called upon to create their own object request-broker implementations on the basis of this specification. Rather than a concrete implementation, CORBA is an abstract specification of the different components, services, and protocols of distributed objects middleware. CORBA-compliant implementations simplify the creation of distributed applications in heterogeneous

environments (Siegel 2000). However, the lack of a reference implementation of this abstract specification, as well as its complexity, has been a major point of criticism against CORBA. Today, there are a number of mature implementations (see commercial implementations at the end of this chapter). Most vendors offer implementations for multiple programming languages and operating systems. To this end, the common specification enables communication between applications that have been created with different programming languages, use different ORBs, and run on different operating systems and in distinct hardware environments. The overall architecture of CORBA is illustrated in Fig. 5.10. It shows that CORBA provides a client-server type of communication. To request a service, a client invokes a method implemented by a remote object, which acts as the server in the client-server model. The line in the middle of Fig. 5.10 indicates the border between the client and the server side.

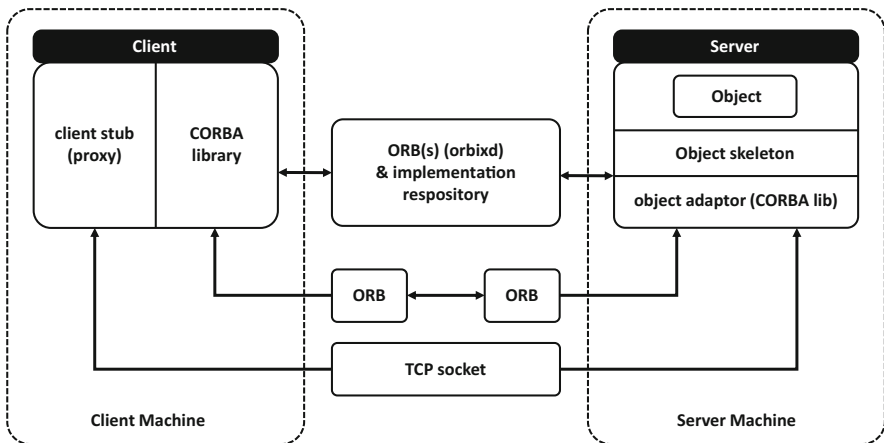


Fig. 5.10 CORBA overall architecture (adapted from Emerald et al. (1998))

CORBA IDL

CORBA IDL, a central component of CORBA, is used to specify the interfaces of objects in a language-neutral way. CORBA IDL builds on the IDL of earlier implementations of RPC and OOM. Like other OOM implementations, CORBA comes with an IDL compiler that automatically generates the client stub and the server skeleton. Furthermore, CORBA defines a standard language mapping for a number of languages like C/C++, Java, C#, Lisp, Smalltalk, Ada, Python, and Objective-C (Baker 1997), which makes it possible to share objects between a number of different programming languages and platforms and makes CORBA an important platform for the development of an assortment of distributed applications (see Fig. 5.11). CORBA also contains components that facilitate the integration of legacy components (e.g., servers implemented using RPC middleware) (Vinoski 1993).

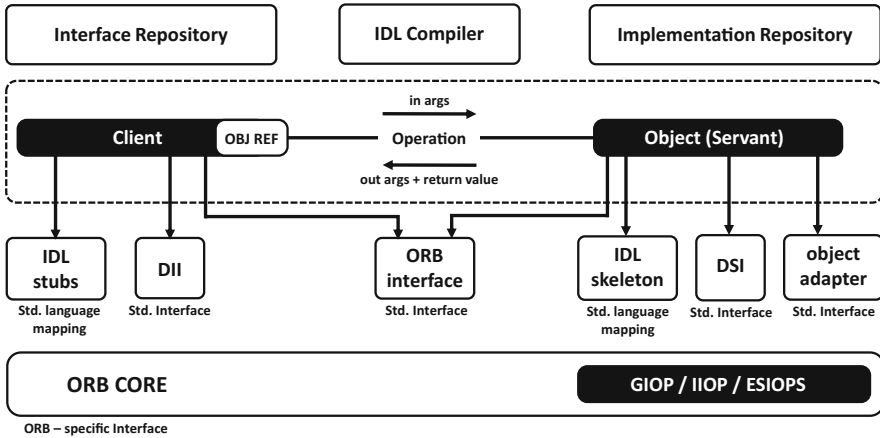


Fig. 5.11 CORBA components

CORBA APIs and protocols

CORBA ensures the portability of client and server code between ORB implementations via a set of standardized programming interfaces (Maffeis and Schmidt 1997). The typical RPC development model assumes that an interface description is used to generate a client stub and a server skeleton during development. However, there are also scenarios in which a client discovers a service at runtime. For this purpose, CORBA contains the Dynamic Invocation Interface (DII), which allows clients to programmatically discover objects and invoke methods without a client stub. In this case, there is no location transparency: The client explicitly constructs and generates CORBA invocations via the DII rather than calling a stub method. The Dynamic Skeleton Interface (DSI) is the server-side counterpart; it can be used to provide services at runtime without a server skeleton (Gokhale and Schmidt 1996). The object adapter is the interface used by the servant object – for instance, to register the implementation, count references, create instances, etc. The Basic Object Adapter contains interfaces specific to the relevant ORB implementation. This means that the server-side code is not portable between different CORBA implementations. For this reason, CORBA introduced the Portable Object Adapter (POA), a specification that standardizes the interfaces used by the object implementation. The term ‘servant’ was introduced for this implementation. The separation between object and implementation allows for very fine control of access on the server side, which is completely invisible to the client. Thus, servants using the POA are portable between different ORB implementations (Pyarali and Schmidt 1998). Standardized APIs ensure that the client- and the server-side codes of a distributed application are portable across different ORBs. CORBA needs to ensure that a client running on ORB can invoke a server running on another ORB. Accordingly, the communication protocol between ORBs (i.e., how method invocations, arguments, and return values are marshaled and what message format is

being used) needs to be standardized. This specification is split into an abstract and a concrete part. The General Inter-ORB Protocol (GIOP) is an abstract specification that contains, among others, details of how to map IDL data types to the wire format. It further describes the format of messages involved in an invocation, how connections are managed, etc. (Vinoski 1997). Internet Inter-ORB Protocol (IIOP) and Environment-Specific Inter-ORB Protocols (ESIOPS) are two of the protocols used in the context of CORBA, and they are based on GIOP.

CORBA evolved at a time when distributed applications were running mainly on local networks. It was originally not designed for communication over the Internet, and using GIOP over the Internet can be challenging because of firewalls (Gokhale and Schmidt 1998). For this reason, some implementations have added protocols that allow ORBs to communicate via HTTP (Object Computing 2019). However, this is not part of the CORBA standard.

Implementation

CORBA is implemented in several different solutions, including The ACE ORB (TAO) from Object Computing Inc. (Object Computing 2019), VisiBroker from Micro Focus International plc (Micro Focus International plc 2019b), and ORBacus, which is also from Micro Focus International plc (Micro Focus International plc 2019a). TAO is a free, open-source, standard-compatible, real-time implementation of CORBA in C++ based on the ACE framework.

TAO offers a scalable quality of service (QoS) for the entire communication path (from end to end). Unlike conventional implementations of CORBA, TAO applies software practices and patterns to simplify the automation of high-performance real-time QoS for distributed applications (Object Computing 2019). The Java-based VisiBroker can be deployed in any Java environment. It offers support for the C++ programming language, object naming, persistent objects, dynamic object creation, and interoperability with other ORB implementations and has the ability to distribute objects across a network (Micro Focus International plc 2019b). ORBacus is a CORBA-compliant middleware that is used on mission-critical systems in the telecommunication, financial, government, defense, aerospace, and transportation industries. ORBacus conforms to CORBA 2.6, is designed for rapid development and deployment, and supports C++ and Java (Micro Focus International plc 2019a).

Commercial Implementations – Component Object Model

The Component Object Model (COM) is a technique developed by Microsoft for inter-process communication in Windows. COM components can be implemented as runtime modules (DLLs) or executable programs. COM is supposed to enable the easy reuse of already written program code, partly also outside the operating system. COM components can be used independently of the programming language (Goos et al. 2001). Microsoft introduced COM in 1992 with the Windows 3.1 graphical user interface. COM is based on the client–server model. A COM client generates a COM component in a COM server and uses the object’s functionality via COM interfaces. Access to objects is synchronized within a process by COM apartments. Many functions of the ‘Windows Platform SDK’ are accessible via COM. COM is the basis on which ‘Object Linking and Embedding’ (OLE) automation and ActiveX

are built. OLE is a low-level, object-oriented technology based on the Component Object Model (COM); it provides services to applications for creating compound documents. ActiveX is a Microsoft software component model for active content. However, with the introduction of its .NET software framework, Microsoft pursued the strategy of replacing COM in Windows with this framework, which it called .NET Remoting. .NET Remoting is built into the Common Language Runtime (CLR), which provides uniform interfaces to other technologies in .NET. Microsoft now considers .NET Remoting an obsolete technology, and it is no longer recommended for the development of distributed applications. The recommended successor technology is the Windows Communication Foundation (WCF), which is also part of .NET.

Commercial Implementations – Windows Communication Foundation

The WCF (formerly codenamed Indigo) is a service-oriented communication platform for distributed applications in Windows. It combines many network functions and makes them available to the developers of such applications in a standardized way. The WCF combines the DCOM, Enterprise Services, MSMQ, WSE, and Web Services communication technologies in a uniform programming interface (Microsoft 2017). The WCF replaces .NET Remoting and can be used in the development of service-oriented architectures. It also enables interoperability with Java Web Services, which have been implemented using Web Services Interoperability Technology (Microsoft 2017).

Commercial Implementations – Distributed Component Object Model

The Distributed Component Object Model (DCOM) is an object-oriented remote procedure call system based on the Distributed Computing Environment standard. It was developed by Microsoft to let the Component Object Model software communicate over a computer network. It was formerly known as Network OLE. These two names indicate the origin of the protocol. It expands the earlier interface definitions, Component Object Model and Object Linking and Embedding, to include networks. DCOM was developed by Microsoft, which is the model's primary user (e.g., with ActiveX) (Microsoft 2018). However, there are also different adapters that make it possible to communicate via a DCOM protocol without directly using Microsoft's DCOM implementation. One example is JCOM, which creates an interface between Java and DCOM (Marin 2003).

Commercial Implementations – Remote Method Invocation

RMI is the call to a method of a remote Java object and implements the Java-specific nature of RPC (Oracle 2019b). Here, 'remote' means the object (and its methods) can be located in another Java virtual machine, which, in turn, can run on a remote computer or on a local computer. The execution of the calling object (or its programmer) looks exactly like a local call, but special exceptions must be handled, such as a terminated connection to a remote computer. On the client side, the stub takes care of the network transport. Before the release of version 1.5.0 of the Java 2 Standard Edition (J2SE), the stub was a class created with the RMI compiler `rmic`. The stub is created by the Java Virtual Machine. Remote objects can also be

provided by a remote object already known in the program, but for the first connection, the address of the server and an identifier (an RMI URL) are required. For the identifier, a name service on the server returns a reference to the remote object. For this to work, the remote object must have previously been registered with the name service on the server under this name. Since version 1.1 of the JDK, TCP/IP-based sockets can be extended with user-defined protocols. The class java.net also supports connections based on BSD sockets (Berkeley Software Distribution). This kind of communication has to be regarded as low-level programming. In addition to a persistence mechanism (serialization), which secures objects and their instances, a remote method layer has been added in order to allow – in line with high-level programming – development at the object or the component level, regardless of the communication on which it is based.

Summary

The term middleware has been in use for over 50 years and the middleware concept as a solution for linking newer applications to older legacy systems already gained wide-spread adoption in the 1980s. Regardless of its long existence, middleware remains highly relevant in today's IT landscape. When developing or integrating new systems, organizations often have to rely on the unique functionality of preexisting applications or entire systems. This chapter presented middleware as a type of software used to manage and facilitate interactions between IT systems and applications across computing platforms.

A well-known example of middleware is the Remote Procedure Call (RPC). RPC is the most basic type of middleware, by allowing functions to be called in other address spaces. The invoked functions and the calling program are usually not executed on the same computer. There are many implementations of this technique and they tend not to be compatible with each other. From a more abstract perspective, the general idea of a procedure call fits the request–response pattern of the client-server model. The client makes a request to some external code, sleeps, and finds the result after control is returned to the procedure. Thus, procedure calls constitute a natural programming abstraction for the request–response message exchange pattern.

Literature provides several categorizations and descriptions of middleware. These different categories result from the long evolution of middleware and its ongoing improvement in commercial implementation. This chapter defined middleware into three main categories. These categories are message-oriented (MOM), transaction-oriented (TOM) and object-oriented middleware (OOM). MOM refers to middleware based on synchronous or asynchronous communication. A MOM system client can send messages to and receive messages from other clients through the messaging system. TOM supports synchronous and asynchronous communication among heterogeneous hosts and facilitates integration between servers and database management systems. TOM is often used with distributed database applications.

One of the oldest and best-known types of TOM is the TP monitor. TP monitors were developed as multithreaded servers to support large numbers of terminals from a single process. The TP monitor is also the most reliable, most stable, and best-tested technology in the enterprise application integration context. The basic idea behind OOM is to make the principles of object-oriented programming (e.g., object identification through references and inheritance) available for the development of distributed systems. OOM is based on the RPC concept and, thereby, enables communication between objects within a distributed system (e.g., the Internet) and is independent of both operating systems and programming languages. OOM supports interoperability between objects and provide location transparency. The most wide-spread OOM standard is the Common Object Request Broker Architecture (CORBA).

In addition to all its many advantages in terms of integration of heterogeneous systems and applications, middleware also bears several drawbacks. One of the biggest drawbacks of middleware is its size and slow performance compared to highly integrated implementations. An optimization of the performance of these programs by the developers is rarely possible.

Questions

1. What is location transparency in the context of RPC?
2. Explain step by step what happens when a client makes a remote procedure call.
3. How do existing middleware categories differ from each other?
4. What are the components of a TP monitor?
5. What is the difference between a local and a remote procedure call?
6. Why do we need middleware when various software components already offer APIs?
7. What are the most common commercial implementations of middleware?
8. What is the difference between a Web service and middleware?

References

- ACM (2017) ACM software system award. <https://awards.acm.org/software-system/award-winners>. Accessed 7 Sept 2019
- Aiken B, Strassner J, Carpenter BE, Foster I, Lynch C, Mambretti J, Moore R, Teitelbaum B (2000) RFC 2768 – network policy and services: a report of a workshop on middleware. <https://tools.ietf.org/html/rfc2768>. Accessed 17 Sept 2019
- Albano M, Ferreira L, Pinho L, Rahman Alkhawaja A (2014) Message-oriented middleware for smart grids. *Comput Stand Interfaces* 38(C):133–143
- Alonso G (2018) Transactional middleware. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, New York, NY, pp 4201–4204

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) *Web services: concepts, architectures and applications*. Data-centric systems and applications, 1st edn. Springer, Berlin
- Amazon Web Services (2019) Amazon simple queue service. https://aws.amazon.com/sqs/?nc1=h_ls. Accessed 1 Apr 2019
- Apache Software Foundation (2019) Apache ActiveMQ. <http://activemq.apache.org/>. Accessed 1 Apr 2019
- Baker S (1997) CORBA distributed objects: using Orbix, vol 7. Addison-Wesley, New York, NY
- Balasubramanian J, Schmidt DC, Dowdy L, Othman O (2004) Evaluating the performance of middleware load balancing strategies. Paper presented at the 8th IEEE international enterprise distributed object computing conference, Monterey, CA, 24 Sept 2004
- Belokosztolszki A, Eysers DM, Pietzuch PR, Bacon J, Moody K (2003) Role-based access control for publish/subscribe middleware architectures. Paper presented at the 2nd international workshop on distributed event-based systems, San Diego, CA, 8 June 2003
- Bernstein PA (1996) Middleware: a model for distributed system services. *Commun ACM* 39(2):86–98
- Bernstein PA, Newcomer E (2009) *Principles of transaction processing*, 2nd edn. Morgan Kaufmann, San Francisco, CA
- Bernstein PA, Goodman N, Hadzilacos V (1995) *Concurrency control and recovery in database systems*. Addison-Wesley, Boston, MA
- Birrell AD, Nelson BJ (1984) Implementing remote procedure calls. *ACM Trans Comput Syst* 2(1):39–59
- Bouchenak S, de Palma N (2009) Message queuing systems. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, Boston, MA, pp 1716–1717
- Capra L, Emmerich W, Mascolo C (2001) Middleware for mobile computing: awareness vs. transparency. Paper presented at the 8th workshop on hot topics in operating systems, Elmau, 20–22 May 2001
- Cattell R (2011) Scalable SQL and NoSQL data stores. *SIGMOD Rec* 39(4):12–27
- Chorafas DN (1998) Functions of transaction processing monitors. In: Chorafas DN (ed) *Transaction management: managing complex transactions and sharing distributed databases*. Palgrave Macmillan, London, pp 111–131
- Clark M, Fletcher P, Hanson JJ, Irani R, Waterhouse M, Thelin J (2002) *Web services business strategies and architectures*. Apress, Berkeley, CA
- Cugola G, Jacobsen H-A (2002) Using publish/subscribe middleware for mobile systems. *SIGMOBILE Mob Comput Commun Rev* 6(4):25–33
- Curry E (2005) Message-oriented middleware. In: Mahmoud Q (ed) *Middleware for communications*. Wiley, Chichester, pp 1–29
- Czaja L (2018) Remote procedure call. In: Czaja L (ed) *Introduction to distributed computer systems: principles and features*. Lecture notes in networks and systems. Springer, Cham, pp 141–155
- Emerald P, Yennun C, Yajnik S, Liang D, Shih JC, Wang C-Y, Wang Y-M (1998) DCOM and CORBA side by side, step by step, and layer by layer. <http://course.ece.cmu.edu/~ece749/docs/DCOMvsCORBA.pdf>. Accessed 17 Sept 2019
- Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. *ACM Comput Surv* 35(2):114–131
- Gokhale A (2009) Request broker. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, Boston, MA, pp 2415–2418
- Gokhale A, Schmidt DC (1996) The performance of the CORBA dynamic invocation interface and dynamic skeleton interface over high-speed ATM networks. Paper presented at the IEEE global telecommunications conference, London, 18–28 Nov 1996
- Gokhale A, Schmidt DC (1998) Principles for optimizing CORBA internet inter-ORB protocol performance. Paper presented at the 31st Hawaii international conference on system sciences, Kohala Coast, HI, 9 Jan 1998

- Goos G, Hartmanis J, Van Leeuwen J (2001) Cooperative environments for distributed systems engineering: the distributed systems environment report. Lecture notes in computer science. Springer, Berlin
- Gray J (1981) The transaction concept: virtues and limitations. In: Gray J (ed) Readings in database systems. Morgan Kaufmann, San Francisco, CA, pp 140–150
- Gray J (1993) Why TP-Lite will dominate the TP market. Paper presented at the 5th international workshop on high performance transaction systems, Asilomar, CA, Sept 1993
- Gray J, Reuter A (1992) Transaction processing: concepts and techniques. The Morgan Kaufmann series in data management systems, 1st edn. Morgan Kaufmann, San Francisco, CA
- Haerder T, Reuter A (1983) Principles of transaction-oriented database recovery. *ACM Comput Surv* 15(4):287–317
- Hapner M (2002) JSR-000914 Java message service API. Oracle corporation. <https://jcp.org/aboutJava/communityprocess/final/jsr914/index.html>. Accessed 1 Apr 2019
- Huang AS, Olson E, Moore DC (2010) LCM: lightweight communications and marshalling. Paper presented at the IEEE/RSJ international conference on intelligent robots and systems, Taipei, 18–22 Oct 2010
- IBM (2019) IBM WebSphere MQ version 7.5 documentation. https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.5.0/com.ibm.mq.helphome.v75.doc/WelcomePagev7r5.htm. Accessed 1 Apr 2019
- Khosrowpour M (2002) Issues and trends of information technology management in contemporary organizations. IGI Publishing, Seattle, WA
- Limprecht R (1997) Microsoft transaction server. Paper presented at the IEEE COMPCON, San Jose, CA, 23–26 Feb 1997
- Maffei S, Schmidt DC (1997) Constructing reliable distributed communication systems with CORBA. *IEEE Commun Mag* 35(2):56–60
- Mahmoud Q (2005) Middleware for communications. Wiley, Chichester
- Marin J (2003) BEA WebLogic server 7.X unleashed. Sams, Indianapolis, IN
- Micro Focus International plc (2019a) Orbacus 4.3.2 documentation. <https://www.microfocus.com/de-de/documentation/orbacus/orbacus432/>. Accessed 1 Apr 2019
- Micro Focus International plc (2019b) VisiBroker. <https://www.microfocus.com/de-de/products/corba/visibroker/>. Accessed 1 Apr 2019
- Microsoft (2016) Message queuing (MSMQ). <https://msdn.microsoft.com/en-us/library/ms711472.aspx>. Accessed 1 Apr 2019
- Microsoft (2017) What is windows communication foundation. <https://docs.microsoft.com/de-de/dotnet/framework/wcf/whats-wcf>. Accessed 1 Apr 2019
- Microsoft (2018) The component object model. <https://docs.microsoft.com/en-us/windows/desktop/com/the-component-object-model>. Accessed 1 Apr 2019
- Muppidi S, Krawetz N, Beedubail G, Marti W, Pooch U (1996) Distributed computing environment (DCE) porting tool. In: Schill A, Mittasch C, Spaniol O, Popien C (eds) Distributed platforms. Springer, Boston, MA, pp 115–129
- Naur P, Randell B (1986) Report on NATO software engineering conference. <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>. Accessed 17 Sept 2019
- Nelson BJ (1981) Remote procedure call. Carnegie-Mellon University, Pittsburgh, PA, May 1981
- OASIS (2019) Advanced message queuing protocol. <https://www.amqp.org/about/what>. Accessed 1 Apr 2019
- Object Computing I (2019) The ACE ORB (TAO). <https://objectcomputing.com/products/tao>. Accessed 17 Sept 2019
- Object Management Group Inc. (2012) About the common object request broker architecture specification version 3.3. <https://www.omg.org/spec/CORBA/About-CORBA/>. Accessed 1 Apr 2019
- Object Management Group Inc. (2019) DDS standard for the IoT. <https://www.omgwiki.org/dds/>. Accessed 1 Apr 2019

- Oracle (2019a) Oracle RDBMS. <https://www.oracle.com/technetwork/database/windows/index-088762.html>. Accessed 1 Apr 2019
- Oracle (2019b) Remote method invocation home. <https://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>. Accessed 1 Apr 2019
- Pivotal Software (2019) RabbitMQ. <https://www.rabbitmq.com/>. Accessed 1 Apr 2019
- Pyarali I, Schmidt DC (1998) An overview of the CORBA portable object adapter. *StandardView* 6 (1):30–43
- Ray C (2009) Distributed database systems. Pearson, Dorling Kindersley, Delhi
- Ruh WA, Maginnis FX, Brown WJ (2002) Enterprise application integration: a Wiley tech brief. Wiley, New York, NY
- Shen H (2010) Content-based publish/subscribe systems. In: Shen X, Yu H, Buford J, Akon M (eds) *Handbook of peer-to-peer networking*. Springer, Boston, MA, pp 1333–1366
- Siegel J (2000) CORBA 3: fundamentals and programming, 2nd edn. Wiley, New York, NY
- Umar A (2004) Third generation distributed computing environments. NGE Solutions, Fort Lauderdale, FL
- Uramoto N, Maruyama H (1999) InfoBus repeater: a secure and distributed publish/subscribe middleware. Paper presented at the ICPP workshops on collaboration and mobile computing, group communications, internet, industrial applications on network computing, Aizu-Wakamatsu, 24 Sept 1999
- Vinoski S (1993) Distributed object computing with CORBA. *C++ Rep* 5(6):32–38
- Vinoski S (1997) CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Commun Mag* 35(2):46–55
- Wang J, Bigam J, Wu J (2011) Enhance resilience and QoS awareness in message oriented middleware for mission critical applications. Paper presented at the 8th international conference on information technology, Las Vegas, NV, 11–13 Apr 2011
- White JE (1976) RFC 707 – a high-level framework for network-based resource sharing. <https://tools.ietf.org/html/rfc707>. Accessed 17 Sept 2019
- Wirsing M, Nivat M (1996) Algebraic methodology and software technology. In: 5th international conference, AMAST '96 Munich, 1–5 July, 1996 proceedings. Lecture notes in computer science. Springer, Berlin

Further Reading

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) *Web services: concepts, architectures and applications*. Data-centric systems and applications, 1st edn. Springer, Berlin
- Bernstein PA (1996) Middleware: a model for distributed system services. *Commun ACM* 39 (2):86–98
- Curry E (2005) Message-oriented middleware. In: Mahmoud Q (ed) *Middleware for communications*. Wiley, Chichester, pp 1–29
- Siegel J (2000) CORBA 3: fundamentals and programming, 2nd edn. Wiley, New York, NY
- Umar A (2004) Third generation distributed computing environments. NGE Solutions, Fort Lauderdale, FL

Chapter 6

Web Services



Abstract

Based on the client-server principle, Web services are software systems that interact with client applications and other services through open Web standards. Consequently, heterogeneous computer systems from all over the world can exchange information, regardless of their hardware configurations, operating systems, and software applications. Web services are, therefore, a very popular approach for facilitating automated intra-organizational and inter-organizational communication. This chapter provides a thorough introduction to the Web service concept and the different associated standards and technologies, such as Simple Object Access Protocol (SOAP), RESTful interfaces, and Web Services Description Language (WSDL). This chapter particularly provides a comprehensive introduction for two important Web technologies on which most Web services are based: The Hypertext Transfer Protocol (HTTP) and the Extensible Markup Language (XML). Then, the fundamental Web service architectural principles are explained and two common Web service variants are explored in more detail, namely RESTful and SOAP-based Web services. Using these example implementations, this chapter concludes by comparing the two Web service variants in terms of their different application areas.

Learning Objectives of this Chapter

This chapter's main learning objective is to help readers understand the Web service concept from different perspectives. Once readers have studied this chapter, they will know, from an application perspective, which purposes Web services serve and how Web services facilitate system integration efforts within and between separate organizations. Moreover, this chapter conveys the differences between the two major Web service variants, namely SOAP-based Web services and RESTful Web services. Consequently, readers can make informed choices about the Web service variant best suited for their particular use case. From a technical perspective, this chapter's learning goal is to help readers understand the general architecture behind Web services and how this architecture functions. Having read this chapter, readers should have a comprehensive knowledge of the two most important technologies on which Web services are built: the Hypertext Transfer Protocol (HTTP) and the Extensible Markup Language (XML). Furthermore, readers will understand how

RESTful and SOAP-based Web services define and propagate their service offerings and how service consumers can communicate with them.

Structure of this Chapter

This chapter about Web services is structured as follows: The first section introduces the fundamentals of the Web service concept, including its use in business-to-business integration. The second section provides a comprehensive introduction for the two most important technologies that most Web services currently utilize: the Hypertext Transfer Protocol (HTTP) and the Extensible Markup Language (XML). The final section addresses the Web service architecture, including the service-oriented architecture design paradigm, as well as internal and external Web service architectures. In closing, the two most common implementations of Web services, namely SOAP-based and RESTful Web services, are closely examined and compared.

6.1 Introduction to Web Services

Web services are software systems that interact with client applications and other services via open standards in order to exchange data. These interactions are based on the client-server principle. The Web service, acting as a server, provides its clients with functionalities (e.g., access to resources and processing operations) in the form of modular encapsulated services. Web services make these functionalities available and discoverable through propagation mechanisms, which usually take the form of machine-readable documents that describe all operations exposed by the service. However, despite its name, a Web service can be deployed on any network infrastructure, not only on the World Wide Web (WWW). Web services are characterized by their high interoperability and extensibility, which is achieved by applying standardized Web technologies like the Extensible Markup Language (XML) and the Hyper Text Transfer Protocol (HTTP). Leveraging these well-established technologies reduces dependencies on proprietary networks, operating systems, and platforms, thus making Web services ideal for connecting software applications that run on a variety of platforms and frameworks (W3C 2004). Furthermore, multiple Web services can be loosely coupled. Consequently, each Web service is easily replaceable, as the old and new services provide interfaces that are built around the same standardized Web technologies. Hence, it is not unusual that several simple services are chained together to provide advanced value-added services (e.g., e-commerce Web services could integrate external payment, warehousing, and shipping services). The loose coupling also allows Web service interfaces to evolve over time without disrupting the service consumers' ability to communicate with them. In contrast, a tightly coupled system entails a closely intertwined client and server logic, implying that changes to one interface require the other to be updated. Therefore, using a loosely coupled architecture frequently leads to a more manageable software system and enables easier integration into various systems.

Web Service

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, and invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web-based (Mohamed and Wijesekera 2012).

Utilizing Web services successfully depends, to a certain extent, on the application area (Alonso et al. 2004; Sturm and Sunyaev 2019). Web services are a type of distributed system in which service providers and service consumers are situated in different physical locations and communicate over a network. The distributed system's nature is such that communication with a Web service is slower and less reliable than communication on a local system. This has substantial architectural consequences, requiring infrastructure and software engineers to allow for unpredictable network latencies, concurrency issues, and service disruption risks (Waldo et al. 1994). Web services are consequently best suited for applications (1) that have to be accessed over a network, (2) for which communication reliability, speed, and latency are not essential, (3) for which it is impossible to manage deployment updates such that all clients and providers are updated simultaneously, (4) that are split into different system components which are operated on different platforms and vendor products, and (5) for which an existing application is required for utilization over a network and which can be packaged as a Web service (W3C 2004).

One of the most important Web service application areas is cross-organizational data exchange, which is of particular interest to Business-to-Business (B2B) integration (i.e., the integration of heterogeneous information systems between two or more business partners). Owing to the increasing demand for automatized information exchange in modern supply chains, B2B integration has become a necessity in today's economy. For long, the IT systems of different businesses tended to be badly integrated. In the 1990s, sending orders via phone or fax to external suppliers was still commonplace. While the fax interface was often integrated into the ordering party's internal IT systems (e.g., automatically sending an order fax to the business partner), processing an incoming order required manual work. Moreover, fax is mainly popular, because it was for long, and partly still is, an important B2B communication channel. Businesses changed in the late 1990s and 2000s when the Internet and Web technologies spread and were widely adopted. Suddenly, digital communication (e.g., via e-mail or the WWW) had commonly accepted standards. However, processing e-mail orders still involves manual work. Moreover, many companies offered the possibility of placing orders via Web-based online shops. In these cases, a business partner can automatically process orders, whereas the ordering entity must still place orders manually using a Web browser. The latter aspect (automated exchange of data between applications) is similar to Enterprise Application Integration (EAI), which traditionally relies on middleware platforms. However, one of the main differences between B2B integration and EAI is that EAI

integrates applications and systems within an organization (internal perspective), whereas B2B integration interconnects different organizations (external perspective) (Poduval and Todd 2011). This shift from an internal, controllable, and relative heterogeneous IT environment to an external, dynamic, and multi-stakeholder environment is one of the main reasons conventional middleware platforms, which are tailored to match the requirements of each connected system exactly, are less viable for B2B integration (Poduval and Todd 2011).

A simple approach for achieving full B2B integration could be to introduce a common message bus which interconnects all business partners. Such a B2B service bus would resemble a conventional middleware platform hosted by a third party. The advantages would involve a loose coupling between business partners. However, realizing such a B2B service bus is unrealistic for multiple reasons. First, all partners would have to agree on a conventional middleware solution or standard. Second, the B2B service bus would also require a central provider that hosts the mediating infrastructure and which must, therefore, be trusted by all partners with regard to reliability and security. Finally, an enterprise and its business partners would make themselves dependent on this central infrastructure provider (Benlian et al. 2018). Although there are alternative solutions to achieve a full B2B integration, such as establishing direct communication channels between any two business partners (the customer/consumer and the provider), this would mean that for each channel, the two participating partners must first decide on mutual middleware protocols and infrastructure, which introduces dependencies and additional management overhead on both sides.

Web services mitigate these dependencies by encapsulating a provider's internal IT system operations and resources in self-contained services. These services are then made accessible via standardized interfaces that are built around well-established Web technologies, which can be understood by a large number of clients. These interfaces hide the underlying IT infrastructure's actual complexity by abstracting and exposing only the relevant operations and resources. Moreover, these interfaces are usually well-documented and machine-readable, which facilitates their discovery and invocation by automated clients. All of this results in a loose coupling between business partners, who may act as service providers and service consumers, and enables seamless data exchange with hardly any coordination effort. Web services are, therefore, particularly suitable for integrating enterprise IT systems in the B2B context and in similar application areas in which multiple heterogeneous IT systems operated by the same or different parties need to exchange data (e.g., e-commerce platforms cooperating with payment providers and shipping companies).

To better understand the inner workings of Web services, the remainder of this chapter focuses in more detail on the Web service concept's technical and architectural foundations, which includes an introduction to the most important Web technologies commonly used by current Web service implementations, the architectural constructs that enable Web service interactions, and an explication of two common Web service implementations.

6.2 Basic Web Technologies

As stated, standardization is a key concept in the case of Web services. The Web service's main purpose is to support interoperable machine-oriented and application-oriented interactions over a network. These interoperable interactions require interfaces that are known and understood by the Web service and all of its clients. Therefore, Web services are usually based on well-established Web technology standards that are widely understood via various client applications and systems, which can invoke a service with little to no customization efforts. These established Web technologies mainly include the Hyper Text Transfer Protocol (HTTP) and Extensible Markup Language (XML), which will be introduced in the following sections.

6.2.1 *Hyper Text Transfer Protocol (HTTP)*

HTTP is probably the most important technological foundation of the WWW and, in particular, the most commonly used communications protocol for interacting with Web services. HTTP is a simplistic, text-based request-response protocol for distributed information systems (Fielding and Reschke 2014). In 1991, HTTP was originally proposed and a first version, HTTP/1.0, was officially standardized in 1996 (Berners-Lee et al. 1996). The latest version, HTTP/2, was published in May 2015 (Belshe et al. 2015). As a communication protocol, HTTP provides a set of rules that allow two or more entities to exchange data over a computer network. This data is encapsulated in a message format. An HTTP message consists of a start line, an optional message header, and an optional message body (Fielding and Reschke 2014). The starting line defines either the requested method or response status. The message header consists of name-value pairs that describe additional parameters and arguments needed for the data transmission (e.g., the preferred response language or the date on which a message was sent). The message body carries the data payload, which, in the case of a Web service interaction, could contain parameters that are necessary to invoke a particular service operation.

HTTP messages can be sent either as cleartext (i.e., its header and body are visible to any device relaying the message within the network) or enveloped in an encryption layer by using the HTTP extension HTTP Secure (HTTPS). HTTPS allows encrypting the entire communication between a message sender and receiver with the help of the secondary cryptographic protocol TLS (Transport Layer Security). HTTPS is often employed when sensible data is exchanged (e.g., login credentials).

Hyper Text Transfer Protocol (HTTP)
The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems (Fielding and Reschke 2014).

To give a simple example of how HTTP is used, Fig. 6.1 depicts how a browser application (i.e., a client) retrieves the website with the address <http://example.com/info> from a Web server. Once a user had entered the address in their browser's address bar and hit the enter key, an HTTP request message will be sent over a TCP/IP connection to the Web server associated with the domain example.com. Then, the Web server will access the resource identified by `/info` and return the associated Web site data via a HTTP response message.

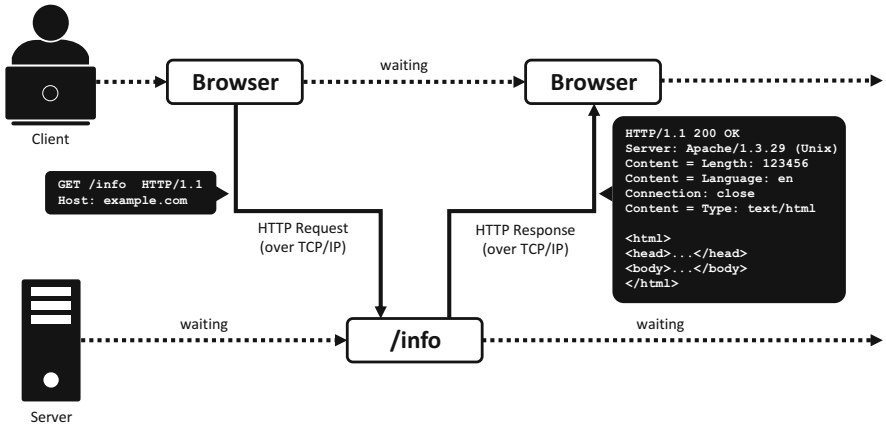


Fig. 6.1 Client-server communication via HTTP protocol

The above example illustrates that there is a distinction between HTTP request messages (sent from client to server) and HTTP response messages (sent from server to client in response to an HTTP Request). In case of an *HTTP Request* message, the starting line specifies which operation the server is supposed to perform. There is a total of nine different HTTP request methods that can be used to retrieve or modify resources. To specify the particular method, a corresponding HTTP request method is used (see Table 6.1) (IETF 2014). In the example request above, the HTTP request method GET is used to request a representation of the resource `/info`.

To specify the resource on which the operation should be performed, a Uniform Resource Identifier (URI) is used. In the case of HTTP, this unique identifier has the

following form: *http://host/path/to/resource?query*. The host part of the URI determines the value of the host header field, which the client will use in the HTTP request. The host-relative path after the host part specifies the requested resource's location on the host's system. Finally, a URI can contain a query, which is indicated by a question mark after the path component. This query part, which can consist of key-value pairs of the form *path?key1=value1&key2=value2*, will be passed as parameters to the resource identified by the path component.

Table 6.1 HTTP request methods

HTTP request method	Description
GET	Request the resource's representation.
HEAD	Identical to GET, but only returns the response message's header.
POST	Post payload data in request body to a resource specified by a URI.
PUT	Store payload data in request body as a resource specified by the given URI.
DELETE	Delete the resource specified by the URI.
CONNECT	Reserved for clients to initiate an SSL tunnel.
OPTIONS	Return HTTP request methods supported by server.
TRACE	Echo the request to check for changes made by intermediate servers.
PATCH	Partially modifies a resource.

Subsequent to a client sending an HTTP Request, the server will process this request and create an HTTP Response. The starting line of an *HTTP Response* contains the protocol version, as well as a status code and message, which indicates the corresponding HTTP request's status. As shown in Table 6.2, the list of status codes is divided into five categories that capture different classes of messages (Fielding et al. 1999). Status codes that begin with a 1 contain informational messages. An example would be if the client's request has triggered a long-lasting operation and the server wants to prevent a time-out on the client's side. In this case, the server can respond with a 102 Processing status, asking the client to wait for the operation to finish. The 2xx category contains different messages indicating the operation's success. The most generic is the 200 OK message, which is, for example, used in response to a successful GET operation. Another example is the 201 Created message, indicating that a new resource was created successfully on the server's side. Status codes in the 3xx category contain redirection messages, which require the client to take further action to complete an HTTP request successfully. For instance, a resource could have been moved to another URI and the client needs to issue a new HTTP request. Finally, the categories 4xx and 5xx indicate errors occurring on the client's or server's side.

Table 6.2 Overview of HTTP status code categories, in accord with IETF (2014)

Status code	Message	Meaning
1xx	Informational	Provisional server responses without body
Examples:		
100	Continue	Indicates that everything thus far is in order and that the client should continue with the request or ignore it if it is already finished
102	Processing	Server is processing a long-lasting operation
2xx	Success	Client request successfully executed
Examples:		
200	OK	Standard response if operation was successful
202	Accepted	The request has been received, but not yet acted upon
3xx	Redirection	Further action required to complete request
Examples:		
301	Moved permanently	Resource moved to another URI
307	Temporary redirect	Directing client to move requested resource to another URI with same method used prior to request
4xx	Client error	Server received erroneous request
Examples:		
403	Forbidden	Client does not have access rights to the content
404	Not found	Resource could not be found
5xx	Server error	Server failed to fulfill a valid request
Examples:		
500	Internal server error	The server has encountered a problem and it does not know how to handle it
504	Gateway timeout	This error response is given when the server is acting as a gateway and cannot obtain a response in time

A key aspect contributing to the HTTP's success and its particular usefulness for certain types of Web services (see Section 6.3.4) is the protocol's support of caching strategies. Caching is one of the key technologies used for building scalable Web services. Without caching, large Internet platforms would not be able to satisfy all incoming HTTP requests; in the case of [Amazon.com](#), for example, these requests are produced by over 2.5 billion visitors per month (Similar Web 2019). The basic idea of caching is to disburden Web servers by introducing intermediary systems (i.e., Web caches), which store previous HTTP responses to answer future HTTP requests. Caching not only helps answer the most common requests quickly, it also facilitates load balancing strategies (i.e., requests are distributed among multiple caching servers). HTTP supports caching by assuming that HTTP GET, HEAD, and OPTIONS requests are cacheable (i.e., a cache system can simply use a cached response to answer the same GET request). The reason for this is that GET, HEAD, and OPTIONS are presumably safe operations, which do not modify the resources they access. Consequently, there is no need to actually pass a GET request to the server, since GET requests can never change the server state. Another operation may

have changed the requested resource in the meantime and in this case, the Web server could proactively invalidate any cached HTTP requests for this resource. Moreover, HTTP provides special header fields that allow for controlling the caching strategy (i.e., how long an answer to a given HTTP request can be cached before it expires). In contrast to GET, HEAD, and OPTIONS, the POST, PUT, and DELETE methods can modify resources and are presumably not cacheable for the reasons outlined above.

6.2.2 Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a data-description language that structures data such that it can easily be understood, retrieved, and shared (W3C 2008). Since XML stores data as plain text instead of proprietary bytecode, data analysis software can be implemented in almost any programming language and be run on almost any computer system. Consequently, XML is widely used for machine-independent, organization-independent, and application-independent data exchange. This high level of versatility has made XML one of the most common formats used for data exchange over the Internet. In the Web services context, XML is often used to describe service functionality, and to format request and response messages sent between the services and their consumers. Table 6.3 depicts a simple XML message, which will be explained in more detail later in this chapter.

XML was developed by the World Wide Web Consortium (W3C) and is available as an open standard. In 1996, XML was first introduced and based on the Standard Generalized Markup Language (SGML), which provided most of XML's foundation. In 1998, the W3C approved XML 1.0 and updated it to Version 1.1 in 2004. XML is extensible to the extent that it allows for defining application-specific data elements along with their structure, order, and how they ought to be interpreted or displayed. XML can also be considered extensible with regard to its nature as a document format; an XML document is a technique for storing data long-term on a local computer or a transient message format enabling communication between two systems (as in the case of Web services). Similar to HTML, XML is a descriptive markup language. Markup describes text added to a document's data that provides information about the document (Goldfarb and Rubinsky 2000). As a markup language, XML defines a set of symbols that can be used to separate, structure, and label parts of a text document. However, XML does not predefine markup, only its syntax and basic interpretation rules. XML, thus, allows to create self-descriptive elements, also called *tags*, that form an application-specific vocabulary and document structure. This makes XML a meta-language, which allows for creating and defining other languages that suit specific application cases. A few examples of XML-based languages are MathML (mathematical markup language), RSS (Web feeds), SPML (organizational service provisioning), XBRL (business reporting language), and, of course, languages for Web service interactions (e.g., SOAP, WSDL, or WS-*).

Extensible Markup Language (XML)

Extensible Markup Language, commonly abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of the computer programs which process them (W3C [2008](#)).

XML is, however, more than merely a markup language, as will be shown in the remainder of this chapter. Instead, XML provides a general and powerful framework to store, exchange, and process structured data. XML can be processed via software that uses standard programming interfaces like XML DOM. Regarding Web services that utilize XML, this kind of interpreters are part of the internal architecture, as described in Section [6.3.2](#).

XML Basic Document Structure

XML documents are text documents that contain only legal Unicode characters. Remember that the Unicode standard specifies a common character set such that it can be understood by computer systems and programmers worldwide. With the exception of a few control characters, all characters specified by Unicode can be used in an XML document. As illustrated by the example in Table [6.3](#), a well-formed XML document starts with a prolog declaration that specifies the information required for interpreting the document, such as the XML version and the document encoding (line 1). The remainder of an XML document forms a hierarchical tree structure consisting of XML nodes (see Fig. [6.2](#)). These nodes always belong to one of four specific types that either represent an XML element, attribute, comment, or text.

- *XML Element*: Markup or content enclosed by a starting and ending tag. XML Elements may have child nodes that are represented via tags, which are markup constructs that begin with an opening angle bracket (<) and end with a closing angle bracket (>).
- *XML Attribute*: Optional key-value pairs embedded in element tags, which cannot have child nodes.
- *XML Comment*: Text ignored by software processing XML documents that allow leaving a human-readable note about the document. XML Comments cannot have child nodes.
- *XML Text*: Contains the text content within elements and cannot have child nodes. Binary data must be additionally encoded (e.g., by using Base64).

The XML tree can include two different node classes: (1) Nodes that may have child nodes and (2) leaf nodes with no other nodes below them in the document structure (e.g., comment and text nodes) (Le Hors et al. [2004](#)). Each non-leaf node can have multiple child nodes, starting with exactly one root XML element at the document's highest hierarchy level.

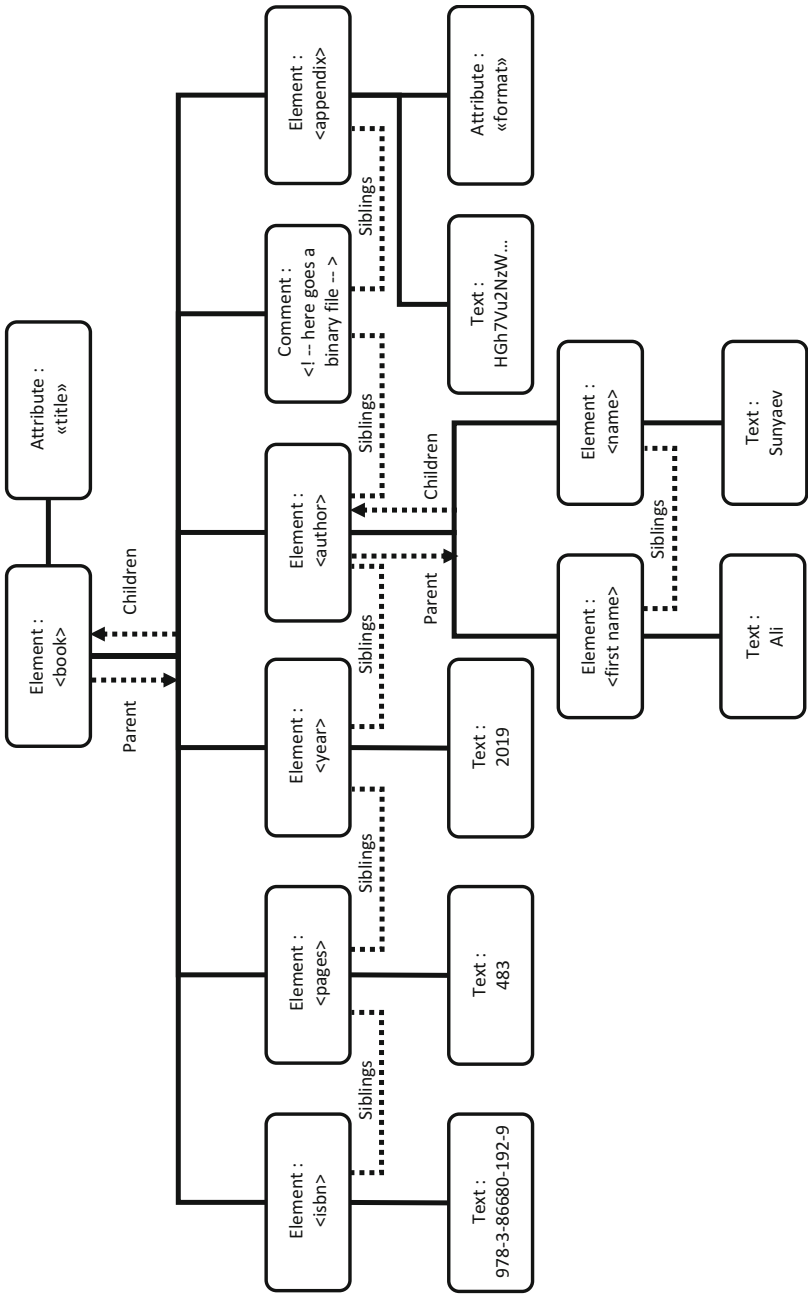


Fig. 6.2 Tree representation of the book example XML document

In the example below (see Table 6.3), the element *book* represents the root of the document. The document's root is declared or opened with the starting tag `<book ...>` (line 2) and closed with the ending tag `</book>` (line 13), which encapsulates the entire document content. XML elements always have a starting and closing tag. If the element has a form of content (e.g., child elements or text), the content is put between two tags in the form of `<tagName> ... </tagName>`. A tag can be opened and closed in the same tag (`<tagName/>`), and, consequently, the node will have no content at this particular moment, but may have defining attributes. If an element has child nodes, the involved tags need to be correctly nested, which means that the tags need to be opened and closed in reverse order. For instance, if element A is opened before element B, B must be closed before A.

An XML element may contain attributes, which modify a tag or help identify the type of tagged information. An attribute consists of a key-value pair (*key=value*) that is embedded in the starting tag. Each element can have only a single attribute with the same name. In the example below (see Table 6.3), the element *book* has an attribute with the key *title* and the value *Principles of Internet Computing*.

Table 6.3 XML Document example

1	<?xml version="1.0" encoding="UTF-8"?>
2	<book title="Principles of Internet Computing">
3	<isbn>978-3-86680-192-9</isbn>
4	<pages>483</pages>
5	<year>2019</year>
6	<author>
7	<firstname>Ali</firstname>
8	<name>Sunyaev</name>
9	</author>
10	<!-- here goes a binary file -->
11	<appendix format="pdf">
12	HGh7Vu2NzW...
13	</appendix>
14	</book>

Namespaces

In the course of processing XML documents, the names of elements and attributes are used to identify and interpret the XML elements' content. However, this only functions if the naming scheme is consistent throughout the entire document. Especially in the Web service context in which XML documents from different services are frequently processed and merged, maintaining a clear mapping between element names and their content is highly challenging. Merging XML fragments can generate name conflicts, particularly in the case of generic identifiers like “name”. For example, two different Web services might use the same name for an XML element while referring to totally different representations in the real world. Particularly while merging the different Web services' output, confusion and misinterpretations could be the result. These conflicts can be avoided by introducing prefixes for elements and attributes. A prefix signals that an element or attribute belongs to one specific namespace that defines a coherent naming scheme. In turn, defining multiple

namespaces within the same XML document allows treating different, but identically named elements, differently.

Prefixes are allocated to a namespace via the reserved *xmlns:<prefix>* attribute of an XML element. Once declared, the prefix can be used in the element in which it was declared, including all children. Namespaces are uniquely identified by a URI that is presumably controlled by the document's creator. A namespace applies to the element in which it is declared and all its children. Elements or attributes are assigned to a namespace via the prefix, followed by a colon and the element's name. Elements and attributes without prefixes are considered part of the default global namespace, even if the element or a parent element defined a specific namespace prefix.

In the following example (Table 6.4), *a:publisher* describes a complex structure with different child elements. *b:publisher*, conversely, only represents the publisher's name, but without any child elements. In this case, both elements describe similar real-world objects: publishing companies. However, an application trying to list all the publisher's portfolio topics described in the document, will encounter an error for the *b:publisher* element, as it does not have a child object that specifies the topics.

Table 6.4 XML Namespace example

1	<?xml version="1.0" encoding="UTF-8"?>
2	<library xmlns:a="http://example.com/a">
3	<a:publisher title="Springer">
4	<a:headquarters>Berlin/Heidelberg, Germany</a:headquarters>
5	<a:uri>http://www.springer.de</a:uri>
6	<a:topics>
7	<a:topic>science</a:topic>
8	<a:topic>humanities</a:topic>
9	<a:topic>technical</a:topic>
10	<a:topic>medical</a:topic>
11	</a:topics>
12	</a:publisher>
13	<b:book xmlns:b="http://example.com/b">
14	<b:name>Principles of Internet Computing</b:name>
15	<b:publisher>Springer</b:publisher>
16	</b:book>
17	</library>

XML Document Validation

An XML parser can only correctly process documents that comply with the XML Syntax Rules (e.g., all opened tags are closed, or correct nesting). However, complying with the basic syntactic rules does not guarantee an XML document's validity in a specific usage scenario. Specifying additional rules for the document's structure (i.e., the document tree), could make sense in the context of a specific application. Web services that use XML to format service requests are particularly reliant on such

rulesets to ensure that the clients' requests are correctly interpreted and executed, as described in more detail in chapter 6.3.3.

In the XML document example in Table 6.3, enforcing that the root element is a *book* element, that the root element contains the child elements *isbn*, *pages*, and *year* in this specific order, and that an author has a first and last name, etc., might make sense. Documents that satisfy these additional user-defined rules are called valid, whereas other documents, such as the *book* element in Table 6.4, which, for instance, lacks the *isbn* element, are called invalid. Importantly, a document that is invalid with respect to such specific rules can still be well-formed in that an XML parser can read it without error. In summary, a valid XML document is always well-formed, but a well-formed XML document is not necessarily valid.

The ability to formally specify such compliance rules for documents is very important, as it allows for defining XML-based data formats. These rules are usually specified in the form of an XML schema. The most widely used language to define an XML schema is the XML Schema Definition (XSD). XSD allows for defining data types and XML document structures. Since XSD is an XML-based language, each XSD document is an XML document in itself, i.e., each XSD document can be processed with the same tool chain and framework used for all other XML documents. In 2001, the W3C published the first version of XSD as a recommendation. The latest version 1.1 was released in 2012.

XML Schema Definition (XSD) Schemata

An XSD schema aims at defining and describing a class of XML documents via schema components to constrain and document the meaning, usage, and relationships of their constituent parts: datatypes, elements, as well as their content, attributes and values (W3C 2012).

As stated, XSD documents are XML documents themselves. Table 6.5 showcases an XSD schema for the XML document example mentioned at the beginning of this chapter (see Table 6.3). All XML elements and attributes used by XSD to specify a schema, are defined in the namespace <http://www.w3.org/2001/XMLSchema>, which is usually assigned to the local prefix *xs:* or *xsd:*. The root of a valid XSD document is always the *xs:schema* element, whose children contain all schema declarations. These declarations usually specify four key components: elements (e.g., tag names), complex element types (e.g., define and restrict data types that can be contained in elements and/or attributes), attributes (e.g., mandatory attributes), and restrictions for values (e.g., a value's maximum length). An XSD schema can be referenced from within an XML document by adding the *xsi:schemaLocation* attribute to the root element. This attribute is declared in the namespace <http://www.w3.org/2001/XMLSchema-instance>, which defines XML elements and attributes that allow for validating XML documents against the XSD schema.

Table 6.5 XSD Schema example

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns="http://example.com"
3    xmlns:xs="http://www.w3.org/2001/XMLSchema"
4    targetNamespace="http://example.com"
5    elementFormDefault="qualified">
6    <xs:element name="book" type="btype"/>
7    <xs:complexType name="btype">
8      <xs:sequence>
9        <xs:element name="isbn" type="xs:string"/>
10       <xs:element name="pages" type="xs:integer"/>
11       <xs:element name="year">
12         <xs:simpleType>
13           <xs:restriction base="xs:integer">
14             <xs:minInclusive value="0"/>
15             <xs:maxInclusive value="2050"/>
16           </xs:restriction>
17         </xs:simpleType>
18       </xs:element>
19       <xs:element name="author" type="atype" maxOccurs="10" minOccurs="1"/>
20       <xs:element name="appendix" type="ftype"/>
21     </xs:sequence>
22     <xs:attribute name="title" type="xs:string" use="required"/>
23   </xs:complexType>
24   <xs:complexType name="atype">
25     <xs:sequence>
26       <xs:element name="firstname" type="xs:string"/>
27       <xs:element name="name" type="xs:string"/>
28     </xs:sequence>
29   </xs:complexType>
30   <xs:complexType name="ftype">
31     <xs:simpleContent>
32       <xs:extension base="xs:string"/>
33     </xs:simpleContent>
34     <xs:attribute name="format" type="xs:string"/>
35   </xs:complexType>
36 </xs:schema>

```

XSD element declarations define the elements' properties, such as the element's name and the target namespace. Furthermore, an element declaration defines the element's type, which constrains the element's potential attributes and children (i.e., simple or complex types). An element declaration can also contain integrity constraints, for example, that particular values must be unique within a certain scope of the XML tree structure (uniqueness constraints) or that values must match another element's identifier (referential constraints). The most basic definition of an element in XSD is that of a simple element, which only contains text content and does not have any attributes. This can be achieved via the XSD element *xs:element*. The name attribute specifies this element's tag name in a document. While, in XML, all data is simply stored as text, XSD enables additional restrictions for the type of data encoded in the element's text content via the type attribute. XSD defines a total of 19 simple data types, such as integer, float, double, boolean, string, date, and time. Whether the node's text content is a valid text representation of the data type specified in the schema, will be checked during the validation process. In the

given example (Table 6.5), the *isbn* element's content is defined as a string value. If the type was defined as a date, the validation of the document would fail, because the elements in the example document contain illegal values – even though the document is well-formed, because the only restriction for XML, generally, is that all values are text.

XSD complex-type declarations are required when XML documents contain more intricate and nested structures that exceed a simple-type element's capabilities. Complex-type elements are defined within the XSD element as *xs:complexType*. Like the simple element, the element tag's name is defined via the name attribute. Furthermore, a complex type may define the child elements' structure and content, which can be of a simple or complex type. In the example (lines 7-23), the complex type *btype* defines an ordered sequence of simple (e.g., *isbn*) and complex (e.g., *author*) child elements. By specifying a sequence, the elements must occur exactly in the shown order. How many times each element can occur in the sequence via the child elements' *minOccurs* or *maxOccurs* attributes, can also be specified. According to Table 6.5, the book element is only valid if it contains between one and ten author child elements. Furthermore, a complex-type declaration may define a list of different child elements from which either one can occur. The complex type also provides the *xs:all* tag, which specifies that each element in the list of elements can occur either once or not at all in any order. Finally, the *xs:any* tag allows for defining extensible schemas that can include arbitrary XML fragments not specified in the schema.

XSD attributes are declared using the *xs:attribute* element. Since the simple-type elements cannot have attributes, an XSD attribute is always an *xs:complexType* element's child. Hence, an element with attributes, but no child elements, must be defined as a complex type. The *name* and *type* attributes of the *xs:attribute* element specify the key and value of the attribute in the element. The value of an attribute can only be of a simple type, i.e. a string interpreted as one of the 19 simple types defined by XSD. Attributes are by default optional and can be omitted without making the document invalid. A mandatory attribute requires adding the key-value pair *use="required"* in the *xs:attribute* element (see Table 6.5, line 22). If an otherwise simple data element requires attributes, the element can be defined as a complex element with one *xs:simpleContent* child that extends the desired simple-type element (*xs:extend* with the simple type as *base* attribute value) by adding an *xs:attribute*. In the provided example, the simple type *xs:string* is extended with the attribute *format*, to allow specifying a file format of the binary data stored in the *appendix* element (see lines 30-35).

XSD-type restrictions enable specifying acceptable values for XML elements or attributes. Preventing arbitrary values for a given type in a valid XML document is occasionally necessary. This can be accomplished by restricting the set of legal values by adding *facets* to *xs:simpleContent* or *xs:simpleType* elements. The *xs:simpleType* elements can be used to define a new data type based on an existing simple type with additional facet restrictions. Facets can, for instance, be used to specify inclusive or exclusive minimum values (*xs:minInclusive* or *xs:minExclusive*) and inclusive or exclusive maximum values (*xs:maxInclusive* or *xs:maxExclusive*).

In the XSD example schema, the element `year` is only valid if its content is an integer value between 0 and 2050 (see Table 6.5, lines 13-16). Other restrictions may set a minimum, maximum, or fixed number for the values' lengths (*xs:length*, *xs:minLength*, or *xs:maxLength*), specify the accepted values' lists (*xs:enumeration*), and set arbitrary regular expressions that values must match (*xs:pattern*).

Once stored in an `.xds` file, an XSD schema can be referenced by XML documents to enable these documents' automatic validation against the schema's rules when processed. Consequently, the document's root element needs to declare the default XML schema-instance namespace (*xmlns:xsi*="<http://www.w3.org/2001/XMLSchema-instance>") and specify the XSD schema file's location using the *schemaLocation* attribute (*xsi:schemaLocation*="http://example.com/book_schema.xsd"). The XSD document and referencing XML documents can now be passed to special XSD validation tools, i.e. ready-made tools that take the schema definition and validate whether the document complies with the schema. Schema validation functions are included in all major XML frameworks and the validation step is usually the first step in the processing of an XML document. A successful validation guarantees that the XML tree has the expected structure, based on the schema. Many XML editors have integrated live validation, i.e. while typing, these XML editors show which elements violate the schema referenced in the document.

6.3 Web Service Architectures

In the following, the Web services' architectural foundations are explained, including the service-oriented architecture design pattern, as well as the internal and external Web service architecture perspectives. To exemplify how these foundations are applied in practice, this chapter explores and compares two wide-spread Web service variants, namely RESTful and SOAP-based Web services.

6.3.1 Service-Oriented Architecture

The Web service concept is closely tied to a system design paradigm called service-oriented architecture (SOA), which can be implemented through Web service technologies. The paradigm essentially aims at increasing the reusability of business processes by encapsulating these processes and their sub-processes in individual, automated services that can be integrated by more than one client application (Alonso et al. 2004; Papazoglou 2012). Consequently, SOA describes an abstract information system architecture that delivers services to clients over exposed (i.e., published and discoverable) interfaces. Clients can either be applications that directly consume the provided resources or other services that compose different resources into new service offerings. SOA essentially defines services, the associated components, and the IT infrastructure that makes it possible to compose applications

from services written in different programming languages, provided by different producers, or deployed on various hard and software platforms.

A service in the SOA context can be described as a “representation of a repeatable business activity that has a specified outcome, that is self-contained, that may be composed of other services, and that is a ‘black box’ to consumers of the service” (The Open Group 2009). In practice, this definition usually translates into a piece of software that performs a repeatable activity with a specified outcome. In order to facilitate their integrability, the behavior of these services needs to be self-descriptive with regard to the required input and expected output. For instance, a credit card company may provide a service that checks a credit card number's validity. The exposed service description would specify an input for the service, i.e. a credit card number, and the output would be a simple assessment of either valid or invalid. Which database is accessed or how the retrieval process is implemented, is neither documented, nor of particular interest to the service consumer. That the service has a predictable outcome for a particular input is the only thing that matters (Alonso et al. 2004; Papazoglou 2012). Accordingly, unlike the exposed service interfaces, the underlying service activity's actual implementation usually remains a black box, i.e. inputs and outputs are observable, while the inner workings are hidden. Besides a predictable functionality, services are not restricted with regard to their granularity. A service may perform a simple business function or a highly complex process that consists of multiple other services. However, the underlying activity's modularity is important. This means that the activity is of use to multiple clients or in different application cases, such that it can be implemented as a self-contained, repeatable piece of software. For instance, validating a credit card number may also involve checking whether the customer associated with the credit card number has valid bank account details, which requires requesting information from another system.

Service

A service is a logical representation of a repeatable business activity that has a specified outcome, is self-contained, may consist of other services, and is a black box to the service's consumers (The Open Group 2009).

Encompassing the service concept, SOA describes an infrastructure that facilitates discovering and using services while maintaining the loose coupling between service providers and consumers (Huhns and Singh 2005). On a high abstraction level, this infrastructure consists of three components: (1) service provider, (2) service broker, and (3) service requester. The service provider hosts services and exposes interfaces allowing the service requester to access these services. The description of these service interfaces and all other means necessary to access the service are then published on registries referred to as the service broker. A service requester – or even a potential service requester – can now query the broker's repository to find a service offering with matching characteristics. Once a matching

service is found, the service broker passes the required information about the service to the requester, who then can bind the service interface and invoke the underlying service. This segregation of service providers and service requesters is what makes SOA highly service-centered instead of provider-centered. Based on this infrastructure, it becomes possible to have multiple providers offering the same service and to add or replace different service providers that offer the same service while not affecting the service requester. Moreover, service requesters can automatically locate the relevant services and service providers they actually require. This dynamic publish-find-bind triangle described by SOA (see Fig. 6.3), enables constructing highly flexible and reliable IT infrastructures, which is ideal for composing or integrating heterogenous systems (Georgakopoulos and Papazoglou 2009). This dynamic triangle further decreases the effort in providing and consuming services over the Web.

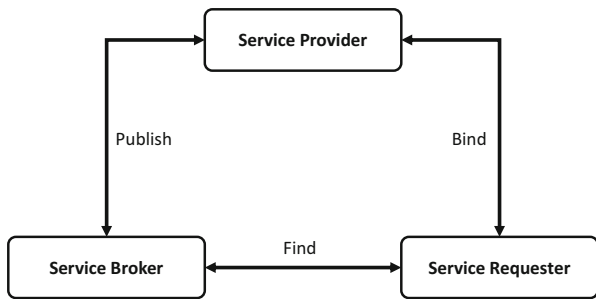


Fig. 6.3 Conceptual interactions within an service oriented architecture (SOA)

Service Oriented Architecture (SOA)

SOA represents a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functions that are built as software components, i.e. discrete pieces of code and/or data structures that can be reused for different purposes (NIST 2013).

Web services, however, do not necessarily have to be deployed in a SOA environment and SOA implementations do not have to be based on Web service technologies. However, while, in theory, SOA could be instantiated using almost any software paradigm, it would be difficult to reach the required degree of decoupling and support from both service providers and service consumers without utilizing the established Web service standards, which, in turn, are based on widespread Web technologies.

6.3.2 Internal and External Web Service Architecture Perspectives

Recapitulating the previous sections, the Web service's main purpose is to expose the internal operations or an IT system's resources such that clients can invoke or retrieve them over a network. Therefore, Web services need the ability to receive inbound requests and to relay these requests to an underlying IT system. There, internal services, applications, and resources are utilized to fulfill the request and produce an outbound response. From an architectural perspective, this short description reveals two perspectives, i.e. internal and an external. As depicted in Fig. 6.4, the internal Web service architecture encapsulates the internal operations and resources described by the Web service interface, while the external architecture facilitates the external clients' access to this interface. The overall Web service architecture can, thus, be described as the combination of two distinct middleware architectures that share a common service interface.

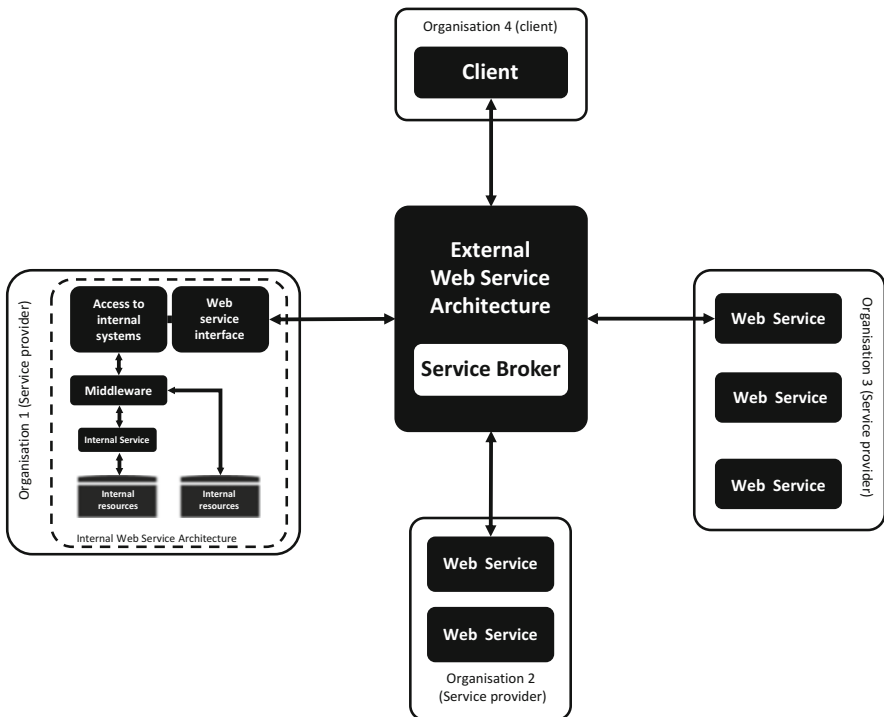


Fig. 6.4 Internal and external Web service architecture (adapted from Alonso et al. (2004))

The **internal Web services architecture** uses conventional middleware concepts for accessing the underlying system's internal operations and resources. Interactions with these operations and resource are usually not performed by the Web services interface, but through subsequent middleware layers. A Web service interface usually only accepts inbound messages. The immediate program logic behind the interface handles packing and unpacking messages, and translates them into a format understood by the adjacent middleware layer (Alonso et al. 2004). This middleware layer either directly accesses the internal operations and resources or interacts with additional service abstraction layers, which may be a composition of higher order services and resources. Put differently, the Web service's internal architecture may describe a complex, multi-level system in which services are stacked atop other services, applications, and resources. This compositional approach closely resembles those of conventional middleware approaches described in Chapter 5. However, from an architectural viewpoint, implementing an internal Web service architecture basically requires an extra level on top of the existing middleware layers to provide the ability to communicate over standard Web protocols and interpret Web message formats. Two examples of how the interface level can be implemented are described in Sections 6.3.3 (SOAP Web services) and 6.3.4 (RESTful Web service).

The **external Web services architecture** facilitates access to the internal services exposed by the Web service interface. Similar to the internal architecture, the external architecture can essentially be described as a middleware acting as a mediator between service providers and service consumers. In order to gain access to a specific Web service, the consumer must transmit a request to the provider. The provider may then accept the request and perform the desired services. Theoretically, this exchange requires only one initial contract between the provider and the consumer that defines the necessary exchange parameters (e.g., service location, available operations, message format). In this case, any additional intermediaries could be perceived superfluous; furthermore, the external architecture may only consist of a protocol infrastructure that coordinates the interactions between and composition of different Web services (e.g., peer-to-peer protocols) (Alonso et al. 2004). However, as described in the previous chapter, SOA provides convincing arguments for additional entities that decouple services from their consumers for increased reliability, agility, and convenience. These entities, which can be considered a part of a Web service's external architecture, may take the form of a central (third-party) service broker (see Section 6.3.1). This service broker's role varies depending on its particular implementation. For example, a service broker can act as a simple repository for service descriptions that only facilitate the service consumers' initial discovery. A more extensive service broker implementation may also act as a message-oriented middleware by mediating all interactions between Web services and service consumers. These service brokers can decrease interdependencies between Web services and service consumers up to a point at which any change to a Web service (e.g., its network location), or even provider outages, can be

compensated without consumer interventions by adapting the broker's program logic (e.g., redirecting requests to a different location or different Web service). On the downside, this service broker implementation can create a single point of failure for an entire service-oriented ecosystem and requires trust from service providers and consumers, especially when confidential information is exchanged (Lins et al. 2018). Alternatively, the central entity can be substituted with a decentralized peer-to-peer infrastructure (Wang et al. 2003; Ragab et al. 2010).

Understanding the difference between an internal and external architecture is important for comprehending the Web service concept and its implementations. As shown above, both architectural perspectives have separate purposes, which result in distinct technological requirements. While certain Web service technologies specifically target internal architecture issues, other technologies are more suitable for implementing external Web service architectures. The seamless interaction between these architectures is normally accomplished by using established Web standards like XML and HTTP.

6.3.3 SOAP Web Services

A SOAP Web service, which is also called a big Web service, uses an XML messaging architecture and message formats based on the SOAP standard. Available functionalities and resources are propagated through machine-readable service descriptions that are based on the Web Services Description Language (WSDL). The SOAP Web service architecture also allows for synchronous and asynchronous interactions between clients and services, and provides additional features that facilitate routing, transaction processing, etc. The messaging protocol SOAP was originally introduced as the Simple Object Access Protocol, which was developed at Microsoft in 1998 and became an Internet Engineering Task Force (IETF) standard in September 1999. Version 1.2 became a W3C recommendation in June 2003. As the protocol's original name suggests, SOAP was planned as a more open successor technology for distributed object middleware (e.g., CORBA). With the release of version 1.2, SOAP is no longer an acronym, but the protocol's actual name. The acronym was abandoned, since SOAP became more complex and can access any type of resource, not only distributed objects as the original name suggested. Fig. 6.5 provides a conceptual depiction of the interlayered SOAP Web service architecture stack.

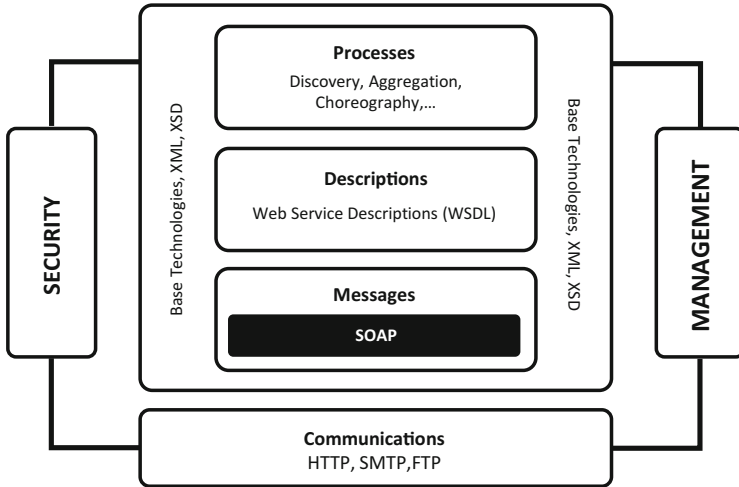


Fig. 6.5 SOAP Web service architecture stack (adapted from W3C (2004))

The SOAP Web service architecture stack illustrates the layering and interrelations between different service concepts and technologies (W3C 2004). Messages are positioned in the center of the architecture, since SOAP Web service interactions are predominantly message-based. The requests to and responses from the Web service follow a standardized message format. These standards include message transport (e.g., HTTP, SMTP, FTP), message encoding (SOAP), and service description (WSDL). Additional standards define means for secure message exchanges (e.g., Web Service Security), service management (e.g., Web Service Management), and service discovery. Next, the architecture's essential aspects are examined in more detail.

SOAP Message Transportation

For the transport of SOAP messages, a variety of network protocols can be implemented. The SOAP specification does not strictly define how messages are exchanged, only how to structure exchanged messages. Consequently, SOAP provides an extensible binding mechanism, which allows it to use any underlying transport protocol. SOAP frameworks usually offer bindings for established protocols, like HTTP, SMTP, and FTP. For the HTTP binding, there is a possibility, at least theoretically, of using the HTTP GET and HTTP POST request methods to exchange messages. However, an HTTP GET means that the GET request does not include a SOAP message, i.e. only the response by the server contains a SOAP message. This is not supported by most middleware platforms. The available bindings are propagated in the description of a specific Web service (i.e., a WSDL file), as later exemplified in Table 6.8.

SOAP Message Processing

SOAP Web services operate according to a distributed processing model that consists of several components. Clients generate SOAP messages that are sent to Web services. These messages consist of a body and an optional header. En route to its receiver, a SOAP message can pass through multiple intermediaries, which receive, process, and possibly change the message. These intermediaries can be classified into forwarding and active intermediaries. A forwarding intermediary simply relays the message on the path to the final receiver, which can be dynamically determined based on a certain routing strategy. Such an intermediary node can process and change certain headers. By specifying an additional role attribute in the header field, the sender can control which of the intermediaries should process which header. An active intermediary can, furthermore, change the body of a message, which might not have been specified in the message's header. Services that an active intermediary can provide, include logging, content modification, tracing, and security (W3C 2002). For example, an organization might employ an intermediary service that attaches a digital signature header to each outbound message leaving the local intranet to allow receivers to automatically verify a message's integrity (Graham et al. 2005).

As stated earlier, SOAP was originally conceived as a method for remotely accessing distributed objects via XML and HTTP. Consequently, SOAP directly supports synchronous client-server communications similar to the concept of remote procedure calls (RPC) (Thurlow 2009). In distributed computing systems, RPC request the execution of functions provided by a remote software component. Similar to internal function calls within software programs, RPC can supply extra arguments when invoking remote functions and will wait for the remote system to return the requested answer. SOAP Web services can implement RPC-style interactions via messages that contain a single XML element for each method called by the server. This method's child elements then contain the arguments that should be passed to the method, if required. The server answers with a specially structured response message which contains the return value(s) of the method that was called. The message format is implicitly coupled to method signatures and different middleware may encode arguments differently.

Besides RPC-style interactions, the document-style interaction is a second model. Here, as the name suggests, client requests are sent in the form of an entire XML document. The structure within the documents' body is not predefined by SOAP formatting rules. These documents may contain any XML data, as long as the client and the service agreed to it, i.e. the data is validated against the service's XML schema. Furthermore, requests in document-style interactions do not, by default, expect a service to respond. Hence, this style allows for an asynchronous message exchange, which makes it more suitable for building loosely coupled systems. Furthermore, the exchange of messages based on a middleware-independent schema reduces technological dependencies.

SOAP Message Format

A SOAP message, which can be either a client request or a Web server response, is a valid XML document that uses the default *SOAP envelope* (<http://www.w3.org/2003/05/soap-envelope>) namespace. The document's root element is the SOAP envelope, which must be assigned to the eponymous namespace and carries the tag label *Envelope*. The envelope encloses the message's header and body, which are also XML elements qualified by the SOAP envelope namespace.

The SOAP header element (tag label *Header*) is optional and holds application-specific information, like authentication and routing data. If a header element is defined, the header element must be placed immediately below the root element (i.e., *Envelope*) as the root element's first child. The header element's children must be qualified by a namespace. Further reserved attributes can be specified for these child elements. For example, the attribute *mustUnderstand* indicates whether it is mandatory (1) or optional (0) for a message receiver to process a header entry (Gudgin et al. 2007).

The SOAP body element (tag label *Body*) is always mandatory, as it encloses the actual message content. The body includes either the request information required to invoke a specific service (client request) or the data it generated in response (server response). The SOAP body element's immediate child elements do not necessarily have to be qualified by a namespace. However, using unqualified elements is not advisable, since messages' interpretations tend to become more ambiguous (Gudgin et al. 2007). The body content's actual structure depends on the interaction style (i.e., RPC or document) and the individual Web service's specifications as propagated via WSDL.

The SOAP message's final key component is the *Fault* element that informs Web service clients about errors that occurred while transmitting or processing the preceding service request. In case of an error, the Web service's expected behavior is to return a message with exactly one fault element in its body and no additional sibling elements. However, fault elements may appear anywhere in a SOAP message (e.g., the header block), even though there is no guarantee that an arbitrary client will recognize the message as erroneous (Gudgin et al. 2007). The SOAP fault element has two mandatory child elements: (1) The *Code* element provides a machine-readable fault code and (2) the *Reason* element provides more detailed human-readable information about the fault. Optional elements can provide further information, for instance, about the network node in which the error occurred (*Node*) and application-specific error information (*Detail*).

In the example below (Table 6.6), a client sends a message to the SOAP Web service Library. In this message, the client requests the name of the first person listed as the author of the book with the ISBN 978-3-86680-192-9 by calling the operation *GetFirstAuthor*. This function's implementation is transparent to the client and could, for instance, involve the Library Web service to invoke additional services. After the server processed the request and a matching book is found, an author element with the name of the first author is returned to the client (Table 6.7). In its HTTP header, the response contains a status code that indicates, in this, case success, i.e. 200 OK.

Table 6.6 Example of a SOAP HTTP Request

1	POST /Library HTTP/1.1
2	Host:example.org
3	Content-Type: application/soap+xml; charset=utf-8
4	Content-Length: 1234
5	
6	<?xml version="1.0"?>
7	<soap:Envelope
8	xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
9	<soap:Body xmlns:lib="http://www.example.org/library">
10	<lib:GetFirstAuthor>
11	<lib:ISBN>978-3-86680-192-9</lib:ISBN>
12	</lib:GetFirstAuthor>
13	</soap:Body>
14	</soap:Envelope>

Table 6.7 Example of a SOAP HTTP Response

1	HTTP/1.1 200 OK
2	Content-Type: application/soap+xml; charset=utf-8
3	Content-Length: 1234
4	
5	<?xml version="1.0"?>
6	<soap:Envelope
7	xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
8	<soap:Body xmlns:lib="http://www.example.org/library">
9	<lib:GetFirstAuthorResponse>
10	<lib:Author>Sunyaev, Ali</lib:Author>
11	</lib:GetFirstAuthorResponse>
12	</soap:Body>
13	</soap:Envelope>

SOAP Service Descriptions (WSDL)

In order to be understood and correctly used, a SOAP Web service requires, to a certain extent, an interface description. For instance, a Web service that can take orders by accepting SOAP messages, needs to propagate and explain its functionality to customers and potential customers such that they can use this service. Otherwise, the customers might not know where the service is located or what type of messages, and in which precise format, the Web service accepts. While this could be a verbal description, a standardized or even machine-readable description would be much more versatile and easier to integrate into existing system workflows. SOAP Web services, therefore, usually provide a self-description using WSDL.

The basic idea behind WSDL is that it should not be necessary for clients to have any knowledge about the Web service's actual implementation. The only thing that clients need to know is what sort of messages can be sent and what sort of responses can be expected. This implies that all details of the message format, as well as the mechanisms through which the Web service can be accessed, i.e. the underlying protocol and the location of the service, must be specified. Accordingly, WSDL is

more complex than other interface description languages, such as the Interface Description Language (IDL).

A WSDL file is an XML document that lists and explicates the individual services that are provided by a Web service. A service description comprises the service's name and specific location, the expected request message format, and the service response's specification (W3C 2007). As shown in the example in Table 6.8, a WSDL document's root is a single *description* element, which specifies the default namespace <http://www.w3.org/ns/wSDL>. Underneath this root element, WSDL 2.0 specifies four interconnected child elements (i.e., components), which are depicted in Fig. 6.6.

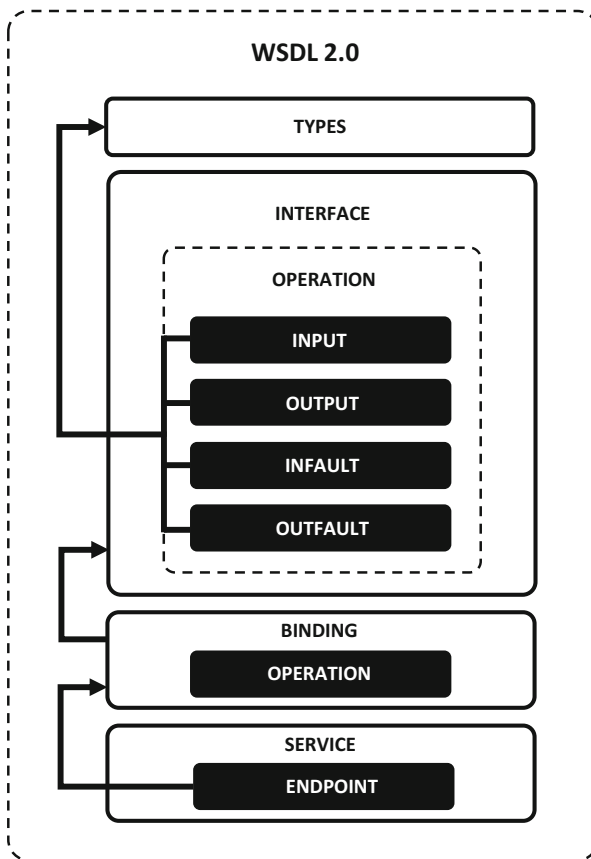


Fig. 6.6 WSDL 2.0 component model

The *types* component describes the complex datatypes a Web service understands and uses to exchange data with its clients. By strictly defining all data types, WSDL documents ensure that data sent or received are correctly interpreted by the sender and the receiver. Since WSDL documents are XML-based, simple data types like

string, integer, and boolean can be referenced directly anywhere in the document. More complex data types, like structures and sequences, must be defined in the `<types>` section, for instance, with the help of an XSD schema. This schema may be specified directly in the types section (inline) or by referring to a separate XSD file. In case a Web service uses only simple data types, the type section may be omitted from the WSDL document.

The **interface** component describes all operations that a Web service offers to clients. The individual operations are defined as the `<interface>` component's child elements. Each `<operation>` element comprises one or more messages, which are grouped into inbound client requests (`<input>`), outbound service responses (`<output>`), as well as incoming and outgoing fault messages (`<infault>` or `<outfault>`). In order to be referenced and invoked, operations require a unique identifier that is defined by its *name* attribute. Extra attributes can be used to specify the operation's behavior. For example, the `<operation>` element's *pattern* attribute defines which message exchange pattern is used: *in-only* (the operation only receives requests), *robust-in-only* (the operation receives requests and can send responses if an error occurred), or *in-out* (the operation receives requests and sends responses). The actual message structures are described as references to datatypes, which may be defined, for instance, in the types-component.

The **binding** component describes the necessary technical details clients require to access interface operations. As stated, SOAP provides an extensible binding mechanism that allows for specifying the message transport protocol. Consequently, three attributes of a `<binding>` element, i.e. *interface*, *protocol*, and *type*, create a reference link between a previously defined interface, a transfer protocol, and a message format type. SOAP Web service interfaces are usually bound to HTTP and use SOAP as message format. In order to assign a specific format to an operation's specific message, additional *operation* elements can be included within the binding component. For example, these elements' *mep* attribute can specify an operation's particular message exchange pattern. Lastly, the binding component allows for defining faults at interface level, which allows assigned operations to reuse the same common error messages.

Finally, the **service** component specifies the service's name (attr. *name*) and defines endpoints. Endpoints are the service component's child elements that associate a particular interface binding (attr. *binding*) with a URI (attr. *address*). Clients can then use this URI to access all operations described by the associated interface.

The following example shows a WSDL document that defines and exposes the `getFirstAuthor` operation used in Table 6.6. In the example, a document-style interaction is defined. This interaction can be initiated via an HTTP client by sending an XML document containing an ISBN element to the URI <http://example.com/libraryservice>. In response, the Web service will return an *Author* name.

Table 6.8 Example for a web service description using WSDL 2.0

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <description
3      xmlns="http://www.w3.org/ns/wsd1"
4      targetNamespace="http://example.com/library"
5      xmlns:lib="http://example.com/library"
6      xmlns:slib="http://example.com/library/schema"
7      xmlns:soap="http://www.w3.org/ns/wsd1/soap"
8      xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
10 <types>
11   <xs:schema
12       xmlns:xs="http://www.w3.org/2001/XMLSchema"
13       targetNamespace="http://example.com/library/schema"
14       xmlns="http://example.com/library/schema">
15       <xs:element name="firstAuthorRequest" type="xsd:string"/>
16       <xs:element name="firstAuthorResponse" type="xsd:string"/>
17       <xs:element name="serviceError" type="xs:string"/>
18   </xs:schema>
19 </types>
20 <interface name="libraryServiceInterface">
21   <fault name="serviceFault" element="slib:serviceError"/>
22   <operation name="getFirstAuthor"
23       pattern="http://www.w3.org/ns/wsd1/in-out">
24     <input messageLabel="In" element="slib:firstAuthorRequest"/>
25     <output messageLabel="Out" element="slib:firstAuthorResponse"/>
26     <outfault messageLabel="Out" ref="lib:serviceFault"/>
27   </operation>
28 </interface>
29 <binding name="libraryServiceSOAPBinding"
30   interface="lib:libraryServiceInterface"
31   type="http://www.w3.org/ns/wsd1/soap"
32   soap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
33   <operation ref="lib:getFirstAuthor"
34     soap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
35   <fault ref="lib:serviceFault" soap:code="soap:Sender"/>
36 </binding>
37 <service name="libraryService"
38   interface="lib:libraryServiceInterface">
39   <endpoint name="libraryServiceEndpoint"
40     binding="lib:libraryServiceSOAPBinding"
41     address="http://example.com/libraryservice"/>
42 </service>
43 </description>
44

```

WS-* Specifications

SOAP and WSDL provide a flexible XML-based messaging infrastructure and a standard service description method. However, neither SOAP, nor WSDL address higher-level functionality that may be required in complex integration scenarios. For instance, SOAP and WSDL do not describe how SOAP intermediaries should route messages, which header fields they should use, how messages should be encrypted, how to define complex business processes, or how transactions on top of SOAP can be implemented. For these complex application cases there is a family of more than 100 additional Web service specifications, i.e. the WS-* framework. These Web

service specifications all share building on SOAP and WSDL to implement more complex functionality.

For example, WS-Man (Web Services-Management) is a communication protocol specification that enables the management of servers, devices, and applications – all via SOAP Web services. The protocol mainly aims at creating interoperability between management applications and managed resources. WS-Man thereby simplifies the administrative data's access and exchange within IT infrastructures and exposes the common operations that clients will use in systems management (DMTF 2018). Another extension is Web Services Security (WS-S), which enhances SOAP by providing a protocol that applies advanced security mechanisms to Web services (Atkinson et al. 2002). For example, the protocol defines routines for enforcing the exchanged messages' integrity and confidentiality. Consequently, WS-S adds support for different security token formats, like X.509, Kerberos, and Security Assertion Markup Language (SAML). WS-S also enables signing and encrypting XML messages for additional end-to-end security, which becomes particularly important when Web service interactions are handled by untrusted intermediaries. Finally, WS-BPEL (Web Service Business Process Execution Language) provides an additional XML-based language, which can be used to describe the logic that connects several Web services, thereby allowing to implement complex business processes (OASIS 2003).

6.3.4 *RESTful Web Services*

Representational State Transfer (REST) describes an architectural style for designing loosely coupled applications over HTTP. When REST is used as basis for Web services, these Web services are referred to as RESTful. Similar to SOAP Web services, RESTful Web services build on the client-server architecture pattern. Clients send requests to a RESTful Web service and, depending on the requested operation, may receive a corresponding response in answer. In contrast to SOAP Web services, RESTful Web services are inherently stateless and focused on providing basic read and write operations to the resources that a service exposes. A resource can be any coherent and meaningful concept, like structured data, texts, images, services, and collections of other resources. However, clients usually do not interact directly with the resources' actual raw data, but with these resources' representations (i.e., representational states) instead. As the name of the underlying architectural style indicates, the conversion between different representational states is a defining characteristic of RESTful Web services. The service converts the resources into a format that matches the client's capabilities and needs, as well as the resource's characteristics (Fielding 2000). Since the service interface conceals this process, clients cannot, or do not have to, differentiate between the resources' actual formats and their representations, which are exposed by RESTful Web services.

The REST architectural style was introduced in Roy Thomas Fielding's dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures" (Fielding 2000). Fielding describes the following six constraints that define the REST architectural style and thereby help describe RESTful Web services –

Client-server: RESTful systems are built on a client-server architecture, which separates clients and servers and, thus, decouples user interface concerns from data storage and data processing concerns. The resulting loose coupling between system components allows each component to scale and evolve independently. The server provides access to information over an efficient and standardized interface. The client retrieves this information and presents it appropriately to users or other systems. This constraint relies heavily on the provision of a reliable interface between components.

Uniform interface: RESTful Web services provide a uniform communication interface between components. Consequently, component interfaces are typically implemented via URIs, which allow for accessing, creating, and manipulating server resources. These resources (e.g., raw data stored in a database) are exposed through the interface to clients, which may manipulate the resources when presenting them to users. Furthermore, a uniform interface ensures that client requests and service responses are self-descriptive and provide sufficient information to explain how these requests and responses should be processed. The resulting communication simplification between components facilitates understanding interactions between components within a RESTful Web service and between services and clients.

Stateless interactions: Interactions with RESTful Web services are stateless. The server neither stores client-specific context information, nor maintains a persistent session state in the server. Therefore, all context-specific and client-specific information that are required to perform a requested service or to provide requested resources must be included in the request (e.g., contained in the URI, query parameters, message body, or header). For example, when a Web service requires clients to authenticate themselves with a username and password, each client request must include the login credentials.

Cacheable: RESTful systems have the capability to cache server responses in order to improve network efficiency, scalability, and performance, for example, by utilizing HTTP's caching capabilities described in Section 6.2.1. Clients, servers, and all intermediary system components can temporarily store responses if the responses implicitly or explicitly define themselves as cacheable. A response can be cached by a client and reused for consecutive equivalent requests without resending the request to the Web service, as depicted in Fig. 6.7. Consequently, cacheable responses need to be idempotent, i.e. they can be applied multiple times without changing the result, to prevent clients from using inappropriate or outdated resources.

Layered system: RESTful systems consist of two or more component layers, including separate layers for clients and servers. Each component can only interact with the components of the layers immediately above or below it. This constraint facilitates replacing and extending individual components, as resulting changes for

other components will be limited to adjacent layers. As shown in Fig. 6.7, a client does not know whether it is directly connected to a database server or an intermediary layer. This decoupling between components would, for example, allow an intermediary layer to implement additional load-balancing mechanisms to increase system availability without changing the client interface.

Code on demand (optional): RESTful Web services may provide downloadable code. Services can thereby extend a requesting application's functionality via program code that is executed on the client system. Such a code extension may, for instance, provide message decryption and encryption functionalities. Clients could download this extension from the Web service and invoke its functions locally to enable secure end-to-end communication with or over the Web service. However, a RESTful Web service usually cannot rely on the code-on-demand constraint, as the service has little to no control over a client's ability to actually execute the distributed program code.

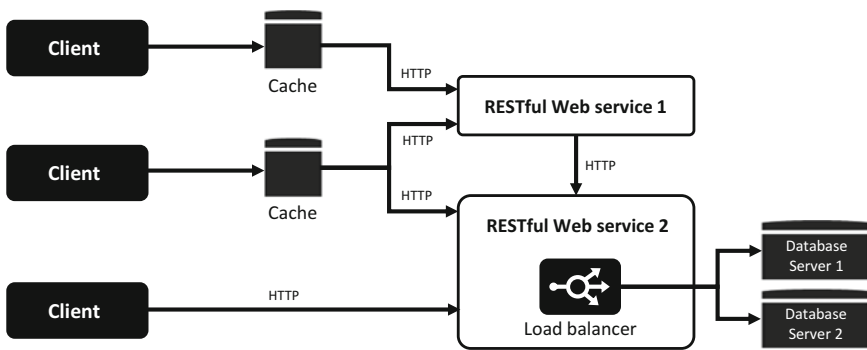


Fig. 6.7 Conceptual architecture of RESTful Web services

RESTful Web Service Interactions

The RESTful Web services' uniform interface design only provides basic operations for accessing and manipulating, i.e. creating, deleting, reading, and updating, resources. Unsurprisingly, HTTP is, therefore, the preferred communication protocol for RESTful Web service interactions. Reconsidering the semantics of the nine different HTTP request methods, the HTTP protocol was actually defined with similar systems, i.e. distributed authoring systems, in mind, which enable creating, reading, updating, and deleting arbitrary resources, like documents and files. A major benefit of the basic set of HTTP operations is that it makes service interactions simpler, more predictable, and idempotent. Consequently, the RESTful Web services' responses that implement an HTTP interface can be easily cached, which, in turn, facilitates the service's scalability and reliability.

The basic idea of RESTful HTTP interactions is that each resource can be identified using a unique URI. The URI is usually of the following format:
`<protocol>://<service-name>/<ResourceType>/<ResourceID>`

To interact with the resource, the client sends an HTTP request to the URI. Which operation the service performs, is determined by the request's HTTP request method specified in its header: create (PUT/POST), read (GET), update (PUT/POST/PATCH), and delete (DELETE). The request's body may contain additional parameters, for instance, data required for creating a new resource, such as a username and password when creating a new user account. The HTTP *accept* header field specifies the desired resource representation in the Web service's response.

The following example in Table 6.9 shows a simple HTTP message that requests a list of books written by the author with the name Sunyaev. Since the desired service operation in this case is *read*, the HTTP request method is, therefore, GET. The resource, i.e. the list of books, is identified by a self-explanatory URI: <http://example.com/books/author/sunyaev>. Furthermore, the request specifies that the resource's representation should be JSON (JavaScript Object Notation), a lightweight alternative to XML that is often used in RESTful Web services interactions (Bray 2017). As requested, the RESTful Web service returns an HTTP response with its payload containing the list of books in JSON notation, which matches the specified resource (see Table 6.10).

Table 6.9 Example of a RESTful HTTP request

1	GET /books/author/sunyaev HTTP/1.0
2	Host: example.com
3	Accept: application/json

Table 6.10 Example of a RESTful HTTP response

1	HTTP/1.0 200 OK
2	Content-Type: application/json
3	Content-Length: 1234
4	
5	[{ uri: "http://example.com/books/isbn/9783866801929",
6	title: " Principles of Internet Computing",
7	year: 2019 },
8	{ uri: "http://example.com/books/isbn/9783834924421",
9	title: " Design and Application of a Security Analysis Method for
10	Healthcare Telematics in Germany",
11	year: 2011 }
12	{ uri: "http://example.com/books/isbn/9783866801929
13	title: " Method Engineering: A Formal Description",
14	year: 2010 },
15	}
16	]
17	

Exposing RESTful Web Services

In order to use a RESTful Web service, clients require knowledge about the available resources, request parameters, data formats, and operations. Consequently, SOAP Web services use WSDL to expose their functionality to clients. The RESTful Web

services' interfaces are, in contrast, usually more simplistic, owing to the limited number of available HTTP request methods they are based on. This results in less need for complex machine-readable service descriptions based on WSDL. In practice, RESTful interactions are often generated and processed by the client via light-weight libraries.

However, exposing a sufficiently large number of resources or providing customized functionalities, could lead to even RESTful Web services requiring more documentation. Consequently, a number of initiatives have begun drafting XML-based languages to describe RESTful Web service interfaces. The most important example is the Web Application Description Language (WADL), which is a machine-readable XML variation for HTTP-based Web services. As shown in the example document (see Table 6.11), WADL describes service resources using HTTP's uniform interface. Similar to WSDL-based service descriptions, clients merely need to consume a service's WADL file in order to access its functionality. In 2009, Sun submitted the WADL specification to the W3C (Hadley 2009). At present, however, the W3C apparently does not intend establishing WADL as an official W3C standard, possibly because it competes with WSDL 2.0. It is not clear which of the competing standards will prevail. Certain arguments seemingly speak in favor of WSDL 2.0, especially since WSDL 1.1 already enjoys wide-spread support. Furthermore, WSDL 2.0 is clearly the language of choice for scenarios in which Web services with both a SOAP and RESTful interface will be exposed. However, the RESTful Web service's WSDL 2.0 description is more complex than, for instance, WADL and it remains to be seen whether parties not using SOAP Web services are willing to adopt it.

Table 6.11 WADL document example

```

1  <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://example.com/wadl.xsd"
3  xmlns:lib="http://example.com/library"
4  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5  xmlns="http://example.com/library">
6  <resources base="http://example.com">
7    <resource path="orderService">
8      <method name="POST" id="orderBook">
9        <request>
10         <param name="isbn" type="xsd:string"
11           style="query" required="true"/>
12         <param name="quantity" type="xsd:int"
13           style="query" default="1"/>
14       </request>
15       <response status="200">
16         <representation mediaType="application/xml"
17           element="lib:OrderResponse"/>
18       <response status="400">
19         <representation mediaType="application/xml"
20           element="lib:Error"/>
21       </response>
22     </response>
23   </method>
24 </resource>
25 </resources>
26 </application>

```

6.3.5 Differentiating Between RESTful and SOAP Web Services

As briefly mentioned in the previous sections and summarized in Table 6.12, there are several major differences between the RESTful and SOAP Web services which need to be considered when making architectural decisions in specific application cases. First of all, the REST architectural style explicitly opposes the idea of stateful distributed objects, while SOAP originally targeted a scenario in which distributed objects are made accessible via HTTP. Even if SOAP is not commonly used in this scenario, it is usually not known whether a service is stateful or stateless, since SOAP makes no assumptions in this respect. Moreover, SOAP primarily uses HTTP POST operations to exchange messages. POST messages are neither assumed to be idempotent, nor cacheable, which makes building scalable Web services more complicated.

Another difference between SOAP Web services and RESTful Web services is that RESTful service requests are usually limited to a uniform set of HTTP operations that are strictly focused on basic data retrieval and manipulation tasks. SOAP Web services can provide an arbitrary set of operations that can be customized for the specific application case. RESTful Web services are, therefore, considered more data-driven, while SOAP is a standardized protocol for transferring structured information, which are more function-driven (W3C 2004).

RESTful Web service interactions usually require fewer network resources (e.g., bandwidth), owing to their better support for lightweight data formats (e.g., JSON). Moreover, RESTful Web service interactions also tend to have a smaller overhead, depending on the interface implementation. SOAP messages tend to have a larger overhead, owing to the envelope around the actual message body. Especially in the mobile computing context, the resulting additional bandwidth requirements could be a relevant factor when deciding on a Web service architecture.

With regard to security, RESTful Web services and SOAP Web services both allow end-to-end transmission encryption via the TLS protocol. Furthermore, SOAP Web services can implement WS-Security, which provides support at the transport level and is more comprehensive than TLS or HTTP over TLS (HTTPS). As a result, SOAP Web services could be more appropriate for integration with common enterprise-level security infrastructures and tools.

Finally, SOAP establishes a close coupling between clients and server, owing to the stringent defined interface (i.e., WSDL specification). On the one hand, this closeness enables the server to exercise more control over interactions. On the other hand, updates and changes to the interface require more adjustments by the clients. By comparison, clients and RESTful Web services are loosely coupled, owing to the light-weight interface between them. While this makes RESTful interfaces less powerful, it also allows for more flexibility when considering the clients' design.

Table 6.12 Differences between SOAP and RESTful Web services

	SOAP Web service	RESTful Web service
Design	Standardized SOAP protocol framework	Architectural style with nonspecific constraints
Approach	Function-driven	Resource-driven
Statefulness	Stateless or stateful	Stateless
Caching	Requests cannot be cached by default	Cacheable
Security	WS-Security with TLS support	Supports HTTPS and TLS
Performance	Large message overhead (SOAP envelope)	Small message overhead
Transfer protocol	e.g., HTTP, SMTP, or UDP	Mainly HTTP
Message format	Mainly XML (SOAP)	e.g., HTML, XML, JSON, or plain text
Pros	High security, standardized approach, extensible functionality	Scalable, requires fewer computing and networking resources, requests cacheable by default
Cons	Requires more network bandwidth, more complex interfaces	Less secure, less flexible, and less functional flexibility

Summary

Web services provide their clients with functionalities in the form of self-contained services that encapsulate the underlying IT system's internal operations and resources. Web services further mitigate dependencies between heterogeneous IT systems via standardized interfaces that allow for exchanging information, regardless of the systems' locations, hardware configurations, operating systems, or software applications. Hence, one of the most important Web service application areas is cross-organizational data exchange, i.e. B2B integration, which can be found, for instance, in the e-commerce context in which vendors are closely tied to external payment providers and shipping companies. Web service standardization is largely achieved via well-established Web technologies, of which two have been detailed in this chapter: HTTP and XML.

HTTP is a text-based, request-response protocol and the most commonly used communications protocol for interacting with Web services. HTTP messages consist of two parts, a header and a body. While the message header contains meta-information needed for data transmission, the message body carries the data payload, which, in a Web service interaction, comprises either parameters necessary to invoke a particular service operation (service request) or an answer from the Web service (service response). An inherent characteristic of HTTP, making it particularly useful to Web service interactions, is the cacheability of service responses, which allows for answering frequent service requests without having to actually invoke the service.

XML is a data-description language that structures data such that it can easily be understood, retrieved, and shared by machines and humans alike. In the Web

services context, XML is often used to describe the functionalities of services and to format request and response messages sent between the services and their consumers. As a meta-language, XML enables creating and defining other languages that suit specific application cases. XML documents are easily processable via off-the-shelf software, as long as these documents comply with the XML Syntax Rules. Furthermore, XML documents must comply with the application context's semantic rules, which are usually specified in the form of an XSD schema.

The Web service design's architectural foundations described in this chapter comprise the service-oriented architecture (SOA) design pattern and the internal and external Web service architecture perspectives. The SOA's fundamental goal is to increase the business processes' reusability by encapsulating these processes and their sub-processes in individual, automated services that can be integrated by more than one client application. SOA describes an infrastructure that facilitates discovering and using services while maintaining the loose coupling between service providers and consumers. In a SOA-ecosystem, the service provider hosts services and exposes interfaces that allow the service requester to access these services. The description of these service interfaces and all other means necessary to access the service are then published on registries called the service broker. From a different perspective, the overall Web service architecture can be described as the combination of two distinct middleware components that share a common service interface, namely the internal and external architecture. The internal Web service architecture encapsulates the internal operations and resources described by the Web service interface, while the external architecture facilitates the external clients' access to this interface.

To illustrate how these architectural concepts are implemented in practice, this chapter discussed two wide-spread Web service variants in detail, namely RESTful and SOAP Web services. SOAP Web services use an XML messaging architecture and message formats based on the SOAP standard. The available functionalities and resources are propagated through machine-readable service descriptions based on the Web Services Description Language (WSDL). The SOAP Web service architecture further allows for synchronous and asynchronous interactions between clients and services, and provides additional features that, for instance, facilitate routing and transaction processing. For more complex application cases, SOAP Web Services can draw on the WS-* framework that implements functionalities, such as advanced security (e.g., WS-S) and management mechanisms (e.g., WS-Man). Conversely, RESTful Web services are based on the Representational State Transfer architectural style, which is defined by six constraints, namely (1) client-server, (2) uniform interface, (3) stateless interactions, (4) cacheable, (5) layered system, and (6) code on demand. The uniform HTTP-based interface design of RESTful Web services only provides basic operations for accessing and manipulating resources (i.e., create, delete, read, and update), which, in comparison to SOAP Web services, makes them more efficient and easier to invoke, but also less flexible. RESTful Web services use the Web Application Description Language (WADL) to expose their functionality to clients.

Questions

1. What is a Web service?
2. What are the design principles behind HTTP?
3. What is the difference between a valid and a well-formed XML document?
4. What is XSD and why is it needed?
5. What is SOAP and what functionalities does it provide?
6. What is the purpose of a WSDL document?
7. Explain how XSD can be used in the context of SOAP-based Web services and WSDL?
8. What are RESTful Web services and how do they differ from SOAP Web services?

References

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) *Web services: concepts, architectures and applications*. Data-centric systems and applications, 1st edn. Springer, Berlin
- Atkinson B, Della-Libera G, Hada S, Hondo M, Hallam-Baker P, Kaler C, Klein J, LaMacchia B, Leach P, Manfredelli J, Maruyama H, Nadalin A, Nagaratnam N, Prafullchandra H, Shewchuk J, Simon D (2002) Web services security (WS-security). <https://msdn.microsoft.com/en-us/library/ms951257>. Accessed 17 Sept 2019
- Belshe M, Peon R, Thomson M (2015) Hypertext transfer protocol version 2 (HTTP/2). <https://tools.ietf.org/html/rfc7540>. Accessed 17 Sept 2019
- Benlian A, Kettinger WJ, Sunyaev A, Winkler TJ (2018) Special section: the transformative value of cloud computing: a decoupling, platformization, and recombination theoretical framework. *J Manag Inf Syst* 35(3):719–739
- Berners-Lee T, Fielding R, Frystyk H (1996) Hypertext transfer protocol – HTTP/1.0. <https://tools.ietf.org/html/rfc1945>. Accessed 17 Sept 2019
- Bray T (2017) The JavaScript object notation (JSON) data interchange format. <https://tools.ietf.org/html/rfc8259>. Accessed 17 Sept 2019
- DMTF (2018) Web services management. <https://www.dmtf.org/standards/ws-man>. Accessed 8 Sept 2019
- Fielding RT (2000) *Architectural styles and the design of network-based software architectures*. Dissertation, University of California, Irvine, CA
- Fielding R, Reschke J (2014) Hypertext transfer protocol (HTTP/1.1): message syntax and routing. <https://tools.ietf.org/html/rfc7230>. Accessed 17 Sept 2019
- Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T (1999) Hypertext transfer protocol – HTTP/1.1. <https://tools.ietf.org/html/rfc2616>. Accessed 17 Sept 2019
- Georgakopoulos D, Papazoglou MP (2009) Overview of service-oriented computing. In: Georgakopoulos D, Papazoglou MP (eds) *Service-oriented computing*. MIT Press, London, pp 1–29
- Goldfarb CF, Rubinsky Y (2000) *The SGML handbook*. Oxford University Press, Oxford
- Graham S, Davis D, Simeonov S (2005) *Building web services with Java: making sense of XML, SOAP, WSDL, and UDDI*, 2nd edn. Sams Publishing, Indianapolis, IN

- Gudgin M, Hadley M, Mendelsohn N, Moreau J-J, Frystyk Nielsen H, Karmarkar A, Lafon Y (2007) SOAP version 1.2 part 1: Messaging framework, 2nd edn. <https://www.w3.org/TR/soap12-part1>. Accessed 17 Sept 2019
- Hadley M (2009) Web application description language. <https://www.w3.org/Submission/wadl/>. Accessed 17 Sept 2019
- Huhns MN, Singh MP (2005) Service-oriented computing: key concepts and principles. *IEEE Internet Comput* 9(1):75–81
- IETF (2014) Hypertext transfer protocol (HTTP/1.1): semantics and content. <https://tools.ietf.org/html/rfc7231>. Accessed 17 Sept 2019
- Le Hors A, Le Hégaret P, Wood L, Nicol G, Robie J, Champion M, Byrne S (2004) Document object model (DOM) level 3 core specification. <https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>. Accessed 17 Sept 2019
- Lins S, Schneider S, Sunyaev A (2018) Trust is good, control is better: creating secure clouds by continuous auditing. *IEEE Trans Cloud Comput* 6(3):890–903
- Mohamed K, Wijesekera D (2012) Performance analysis of web services on mobile devices. *Procedia Comput Sci* 10:744–751
- NIST (2013) Security and privacy controls for federal information systems and organizations. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf>. Accessed 17 Sept 2019
- OASIS (2003) OASIS web services business process execution language (WSBP EL) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. Accessed 17 Sept 2019
- Papazoglou MP (2012) Web services and SOA: principles and technology, 2nd edn. Pearson, Harlow
- Poduval A, Todd D (2011) Do more with soa integration: best of Packt. Packt Publishing, Birmingham
- Ragab K, Helmy T, Hassanien AE (2010) Developing advanced web services through P2P computing and autonomous agents: trends and innovations. Information Science Reference, Hershey, PA
- Similar Web (2019) Amazon.com analytics – market share stats and traffic ranking. <https://www.similarweb.com/website/amazon.com>. Accessed 1 Mar 2019
- Sturm B, Sunyaev A (2019) Design principles for systematic search systems: a holistic synthesis of a rigorous multi-cycle design science research journey. *Bus Inf Syst Eng* 61(1):91–111
- The Open Group (2009) SOA source book. Van Haren Publishing, Zaltbommel
- Thurlow R (2009) RPC: remote procedure call protocol specification version 2. <https://tools.ietf.org/html/rfc5531>. Accessed 17 Sept 2019
- W3C (2002) Relaying SOAP messages. <https://www.w3.org/2000/xp/Group/2/02/27-SOAPIntermediaries.html>. Accessed 17 Sept 2019
- W3C (2004) Web service architecture. <https://www.w3.org/TR/ws-arch/>. Accessed 17 Sept 2019
- W3C (2007) Web services description language (WSDL) version 2.0 Part 1: Core language. <https://www.w3.org/TR/wsdl20/>. Accessed 17 Sept 2019
- W3C (2008) Extensible markup language (XML) 1.0, 5th edn. <https://www.w3.org/TR/xml/>. Accessed 17 Sept 2019
- W3C (2012) W3C XML schema definition language (XSD) 1.1 Part 1: Structures. <https://www.w3.org/TR/xmlschema11-1/>. Accessed 17 Sept 2019
- Waldo J, Wyant G, Wollrath A, Kendall S (1994) A note on distributed computing. https://web.cs.wpi.edu/~cs3013/a11/Papers/Waldo_NoteOnDistributedComputing.pdf. Accessed 17 Sept 2019
- Wang Q, Yuan Y, Zhou J, Zhou A (2003) Peer-serv: a framework of web services in peer-to-peer environment. Paper presented at the advances in web-age information management, Chengdu, 17–19 Aug 2003

Further Reading

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) Web services: concepts, architectures and applications. Data-centric systems and applications, 1st edn. Springer, Berlin
- Moller A, Schwartzbach MI (2006) An introduction to XML and web technologies. Addison-Wesley, Boston, MA
- Papazoglou MP (2012) Web services and SOA: principles and technology, 2nd edn. Pearson, Harlow
- Pautasso C, Wilde E, Alarcon R (2014) REST: advanced research topics and practical applications. Springer, New York, NY
- The Open Group (2009) SOA source book. Van Haren Publishing, Zaltbommel

Chapter 7

Cloud Computing



Abstract

Cloud computing is an evolution of information technology and a dominant business model for delivering IT resources. With cloud computing, individuals and organizations can gain on-demand network access to a shared pool of managed and scalable IT resources, such as servers, storage, and applications. Recently, academics as well as practitioners have paid a great deal of attention to cloud computing. We rely heavily on cloud services in our daily lives, e.g., for storing data, writing documents, managing businesses, and playing games online. Cloud computing also provides the infrastructure that has powered key digital trends such as mobile computing, the Internet of Things, big data, and artificial intelligence, thereby accelerating industry dynamics, disrupting existing business models, and fueling the digital transformation. Still, cloud computing not only provides a vast number of benefits and opportunities; it also comes with several challenges and concerns, e.g., regarding protecting customers' data.

Learning Objectives of this Chapter

This chapter provides a brief introduction to the very diverse domain of cloud computing. Besides understanding how cloud computing emerged from technology improvements and innovations in recent decades, students should get to know essential characteristics of cloud computing, as well as of cloud computing service and deployment models. While other chapters in this book have already introduced several important technologies and related concepts, students will learn in this chapter how these technologies are used in cloud computing to form a multi-layered cloud stack. Finally, this chapter will provide information on the benefits and opportunities of cloud computing, while also pointing out the pitfalls and drawbacks of cloud services, such as those regarding safeguarding customers' data stored in the cloud.

Structure of this Chapter

This chapter proceeds as follows: First, a brief history of cloud computing will introduce readers to the topic. Second, the concept and building blocks of cloud computing will be described, major actors in the market will be explained, and cloud computing will be compared against related concepts. Third, essential cloud

technologies and technical layers that together enable the provision of cloud services are presented. Fourth, the chapter takes an in-depth look at opportunities and challenges of cloud services for organizations, as well as at transformative mechanisms of cloud computing that enable truly innovative services and business models. The chapter explains how, ultimately, these services and models lead to fundamental and large-scale innovations that benefit individuals, organizations, markets, and societies. The chapter closes with a discussion of security and privacy challenges in cloud environments, showing how continuous cloud service certifications can help to tackle these challenges.

7.1 An Introduction to Cloud Computing

7.1.1 *The Emergence of Cloud Computing*

Cloud computing is a model for enabling access to computing resources that evolved in information technology and has become a dominant business model for delivering IT infrastructure, components, and applications (Benlian et al. 2018). With cloud computing, a product-centric model for IT provisioning is transformed into a global, distributed, service-centric model, leading to a disruptive shift from IT-as-a-product to IT-as-a-service (Iyer and Henderson 2010; Benlian and Hess 2011). Since its widespread emergence around 2007, cloud computing has changed the way in which IT services are invented, developed, deployed, scaled, updated, maintained, and paid for (Marston et al. 2011). Cloud computing enables individuals and organizations to access IT resources on-demand, from any device, and at any time, as a measured service (Mell and Grance 2011). It lowers the entry point to high performance computing, allowing organizations to leverage computing power for which they have neither the capital budget, nor the operational expertise to acquire (Arasaratnam 2011). While the market currently urges the availability of IT resources on-demand, cloud providers offer an ever-increasing number and variety of services that are built on a shared pool of computing resources and that can elastically scale up to growing computing demands. Market researchers predict the public cloud market will hit USD 240 billion by 2020, compared to USD 42 billion in 2010 (Gartner 2018a). Today, we heavily rely on cloud services in our daily lives, as in storing data (e.g., *Dropbox*, *OneDrive*), writing documents (e.g., *Office 365*, *Google Docs*), managing businesses (e.g., *SAP ByDesign*), and playing games online (e.g., *GamingAnywhere*, *Stadia*). Cloud computing also provides the infrastructure that has powered key digital trends including mobile computing, the Internet of Things, big data, and artificial intelligence. Thereby, cloud computing accelerates industry dynamics, disrupts existing business models, and fuels the digital transformation (Bharadwaj et al. 2013; Hess et al. 2016). Defying initial concerns, cloud computing has become a critical IT infrastructure for almost every aspect of day-to-day living, and it will continue to transform the world we live in on multiple levels and in various ways, as later sections of this chapter will show.

The history of computing makes it clear that the arrival of the cloud computing era is an evolutionary development (Iyer and Henderson 2010). Cloud computing has its roots in the advancement of several technologies, especially in hardware (e.g., virtualization, multi-core chips), Internet technologies (e.g., web services, service-oriented architectures), distributed computing (e.g., clusters, grids), and systems management (e.g., autonomic computing, data center automation) (Voorsluys et al. 2011). The idea of providing computing *as a service* through networks dates back to the late 1960s and became a driving force of the early Internet development (Venters and Whitley 2012; Berman and Hey 2004). In 1969, Leonard Kleinrock, who is known for the ARPANET project, which is considered to be the foundation of the Internet, said: “*As of now, computer networks are still in their infancy but as they grow up and become sophisticated, we will probably see the spread of ‘computer utilities’ which, like present electric and telephone utilities, will service individual homes and offices across the country*” (Buyya et al. 2009; Kleinrock 2005). This vision became reality with the emergence of ‘application service provision’ during the 1980s (Venters and Whitley 2012; Durkee 2010). With this service-oriented model, third-party providers deploy, manage, and remotely host packaged software applications through centrally located servers and deliver them to organizations through a rental or lease arrangement (Iyer and Henderson 2010). Organizations adopted these application services to reduce complexity, improve organizational agility, and minimize costs by paying only for what they use. However, early application service providers often failed, due to insufficient bandwidth and computing power (Susarla et al. 2003).

During the late 1990s the telecommunications industry experienced tremendous investment and entry of new organizations, which led to a large increase in global fiber-optic networking. Such changes dramatically reduced network latency and related costs (Hogendorn 2011). This improvement of networking was coupled with the emergence of technologies and techniques to coordinate the large-scale, on-demand provision of computing resources (Venters and Whitley 2012), achieved by, i.e., drawing on innovations around ‘grid computing’ (Foster and Kesselman 2004), ‘utility computing’ (Bunker and Thomson 2006), and virtualization of hardware (Killalea 2008). For example, grid computing refers to the paradigm of sharing, selecting, and aggregating large-scale geographically-distributed and (possibly) virtualized resources (software, data, computers) depending on availability, capability, cost, and related quality requirements for solving large-scale problems (Foster and Kesselman 2004; Schwiegelshohn et al. 2010). Today, grid computing is used particularly for solving resource-intensive research problems (Kroeker 2011). Section 7.1.6 will provide a detailed comparison of related concepts. Simultaneous to this development, besides advances in networking, technologies, and techniques, organizations such as *Alphabet* (became the parent company of Google), *Amazon*, and *Microsoft* set up large data centers to transfer computing processes from individual computers and private IT infrastructures to large external and public data centers that were accessible via the Internet. Such data centers’ services soon were labeled ‘cloud computing’ (Venters and Whitley 2012).

The evolution of distributed and service-oriented architectures to provide on-demand computing resources via the Internet, and the vast improvements of technological infrastructures, including networks and data centers, ultimately enabled a shift to the cloud (Carr 2008). The cloud provides computing by means of large pools of automated and scalable computing resources and related applications (Cafaro and Aloisio 2011; Cusumano 2010).

7.1.2 Definition of Cloud Computing and its Essential Characteristics

There are numerous definitions and explanations of cloud computing (Marston et al. 2011; Leimeister et al. 2010; Lins et al. 2019a). The definition put forward by the National Institute of Standards and Technology (NIST) has become established as the most commonly accepted definition of cloud computing among experts. According to this definition, cloud computing is a model that enables ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources that can rapidly be provisioned at any time and from any location via the Internet or a network (Mell and Grance 2011). This includes, e.g., access to networks, servers, storage, or applications. Cloud services are rapidly deployed with minimal management effort, little interaction with the cloud provider, and they can be customized to meet the needs of cloud customers.

Cloud Computing

Cloud computing is a model that enables ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources that can rapidly be provisioned at any time and from any location via the Internet or a network (Mell and Grance 2011).

Characteristic features of cloud services, as described below, include service-based provision of IT resources, on-demand self-service, ubiquitous access, multitenancy, location independence, rapid elasticity, and pay-per-use billing.

Service-Based IT-Resources

All cloud offerings can be expressed as a service. A service is based on a contract, commonly referred to as a Service Level Agreement, which defines the functions it offers and commits to upholding certain quality parameters, e.g., ones regarding service availability (Kumar and DuPree 2011).

On-Demand Self-Service

On-demand self-service enables cloud customers to independently and almost immediately provision computing capabilities, such as server time and network storage (Mell and Grance 2011). Notably, this can be done automatically and

without human interaction by the cloud providers. It is thus possible, e.g., to increase or decrease computing, storage, or application licenses customers have obtained, depending on their current needs.

Ubiquitous Access

Cloud services are provided via a broadband network, mostly using the Internet (Mell and Grance 2011). Cloud services utilize standardized communication interfaces and can be used with a variety of devices, including smartphones, tablets, and laptops. Consequently, cloud customers can access any cloud service from any platform or device at any time (Iyer and Henderson 2010).

Multitenancy

Multitenancy is the ability to have multiple customers leverage shared resources (Kumar and DuPree 2011). Thus, the resources that the cloud provider makes available are used simultaneously by multiple cloud customers (Mell and Grance 2011). Physical and virtual resources are dynamically assigned and reassigned as needed to different cloud customers through a multi-client architecture. Sharing computing resources is part of what makes cloud computing economically beneficial (Arasaratnam 2011).

Location Independence

When using cloud services, there is a sense of location independence in that the cloud customer generally has no control over or knowledge of where the provided resources are actually located. However, on a higher level of abstraction (e.g., country, state, or datacenter), it could be possible to specify location (Mell and Grance 2011). Today, organizations with large data sets typically employ experts whose role is solely to know where data elements exist within their databases, as these organizations have to fulfill various data protection and legal requirements (Iyer and Henderson 2010).

Rapid Elasticity

Most organizations plan their IT processing capacity to cope with peak loads, which is why much of this capacity remains idle for large parts of the time (Iyer and Henderson 2010). In contrast, provisioned cloud resources can be adapted and shared more flexibly, in some cases fully automatically, in order to match the resources to the current needs (Mell and Grance 2011). Due partly to such rapid elasticity, cloud customers have the impression that resources are almost unlimited and are available at any time, to any extent. Cloud providers, therefore, need to create solutions that meet cloud customers' quality of service expectations and cope with large capacity increases, including potentially unforeseeable events (Kumar and DuPree 2011).

Pay-Per-Use Billing

With cloud services, the expectation is that cloud customers are charged for the amount of time they actually use the resource or related measures, a system referred to as 'pay-per-use' (Arasaratnam 2011). Thereby, cloud computing eliminates up-front commitment by users and changes the entry barrier for high performance

computing resources by allowing cloud customers to use only what they need for the time they need it. Cloud providers have to implement features that allow efficient service provision, such as for pricing, accounting, and billing (Buyya et al. 2009). Accordingly, metering should be done for different types of service (e.g., storage or processing) and usage needs to be reported promptly, thus providing greater transparency (Mell and Grance 2011).

7.1.3 *The Cloud Service Market*

The cloud service market comprises five major actors: cloud customers, cloud providers, cloud data center operators, cloud auditors, and cloud brokers. Each actor is an entity (a person or an organization) that participates in a transaction or process and performs tasks in cloud computing (Liu et al. 2011). Below, each actor is briefly described.

Cloud Customer

A cloud customer is a person or organization that maintains a business relationship with, and uses the service of a cloud provider; thus cloud customers are cloud providers' principal stakeholders (Liu et al. 2011). In business-to-business contexts, organizations want an overview of service offerings available in the market before they contact cloud providers who have potentially suitable services that could fulfil the organization's requirements. Organizations should negotiate a contractual relationship regarding the provision of the requested cloud service. Typically, such contracts take the form of service level agreements (SLA). SLAs can cover terms regarding the quality of service, security, data protection, and remedies for performance failures (Liu et al. 2011). Nevertheless, a cloud provider's pricing policy and SLA are mostly standardized and typically fixed, unless the organization expects heavy usage and thus is in a better negotiating position. In business-to-customer contexts, individuals can easily access the cloud service and download applications relevant to them by agreeing to standardized SLA. Cloud services for individuals (e.g., students) typically allow for trial versions and are offered on the basis of a freemium business model (meaning that the basic service features are free of charge, whereas payment is demanded for additional (premium) features, services, or virtual goods). The customers can be billed for the services or for premium features in a 'pay-per-use' manner, and are required to arrange payments accordingly.

Cloud Customer

A cloud customer represents a person or organization that maintains a business relationship with, and uses the service a cloud provider offers (Liu et al. 2011).

Cloud Provider

A cloud provider is a legal person or an organization that is responsible for making a cloud service available to interested parties (Liu et al. 2011). Depending on the service and deployment model, a cloud provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software and operating systems, and provisions the cloud services to cloud customers through a network (e.g., the Internet). Cloud providers' activities include service deployment, service orchestration, cloud service management, service maintenance and upgrades, as well as ensuring security and privacy.

Cloud Provider

A cloud provider is a legal person or an organization that is responsible for making a cloud service available to interested parties (Liu et al. 2011).

Cloud Data Center Operators

Cloud data center operators provide the basic IT infrastructure, including server rooms, redundant or backup components, infrastructure for power supply, data communication connections, environmental controls (e.g., cooling, air conditioning, and fire emergency equipment), and various security devices. Currently, the largest data center operators include *Equinix*, *Digital Realty*, *Level 3*, *Telehouse*, *NTT*, and *Global Switch* (Cloudscene 2018). For example, there are more than 427 data centers in Germany, which are mainly located in and around Frankfurt. Data centers feature rich ecosystems and state of the art equipment, which ensures maximum uptime and connectivity to hundreds of Internet service providers, cloud service providers, and customers.

Cloud Data Center Operators

Cloud data center operators provide the basic IT infrastructure, including server rooms, redundant or backup components, infrastructure for power supply, data communications connections, environmental controls (e.g., cooling, air conditioning and fire emergency equipment), and various security devices.

Cloud Certification Authorities and Auditors

A certification authority is an independent party that assesses whether a cloud service fulfills given requirements (Liu et al. 2011). Certification authorities typically employ auditors to endorse a cloud service as compliant with the set certification criteria, such as criteria imposed by the cloud service certification standard *ISO/IEC 27018* (Sunyaev and Schneider 2013; Lins et al. 2019a). During a certification process, auditors perform comprehensive, manual checks (i.e., document reviews, on-site audits, and penetration tests) to assess adherence according to a defined set of

certification criteria (Lansing et al. 2018). If a cloud provider adheres to specified criteria, a certification authority awards a formal written certificate, which allows providers to embed a graphical web assurance seal on their websites. Cloud service certifications typically aim to ensure the availability, integrity, and confidentiality of provisioned cloud services for a validity period of one to three years (Schneider et al. 2014). Nowadays, auditors can use automated monitoring and auditing techniques, as well as mechanisms for transparently providing certification-relevant information that will continuously confirm adherence to certification criteria (Lins et al. 2018; Lins et al. 2016a).

Cloud Auditors

A cloud auditor is a party that performs an independent assessment to determine whether a cloud service fulfills given requirements (Liu et al. 2011).

Cloud Brokers

As cloud computing evolves, integrating cloud services can be too complex for cloud customers to manage (Liu et al. 2011). A cloud customer can request cloud services from a cloud broker, instead of contacting each cloud provider directly. A cloud broker is an entity that manages the use, performance, and delivery of cloud services and negotiates relationships between cloud providers and cloud customers. In general, a cloud broker can provide services in three categories (Liu et al. 2011) explained below.

- Service Intermediation: A cloud broker enhances a given service by improving certain specific capabilities and providing value-added services to cloud customers. For example, a cloud broker can manage access to cloud services, provide identity management, provide performance reporting, enhance security, or handle the billing process.
- Service Aggregation: A cloud broker combines and integrates multiple services into one or more new services (referred to as Multi-Cloud). The cloud broker provides integration capabilities that link the cloud customer and multiple cloud providers, thus enabling data integration and secure data transmission, as well as additional service intermediation capabilities.
- Service Arbitrage: Service arbitrage is similar to service aggregation, excepting that the services being aggregated are not fixed by any agreements with cloud customers. Service arbitrage ensures that a broker has the flexibility to choose services from multiple cloud providers. The cloud broker, for example, can constantly use a performance or security scoring service to measure and select a cloud service with the best score at a given point in time. Consider a cloud customer wanting to store 100 TB of data, who contacts a cloud broker to fulfill this demand. Based on his expertise and position in the market, the cloud broker will select the best or cheapest option to meet this demand.

Cloud Brokers

A cloud broker is an entity that manages the use, performance, and delivery of cloud services, and that negotiates relationships between cloud providers and cloud customers (Liu et al. 2011).

7.1.4 Cloud Computing Service Models

Cloud services are typically divided into three models (see Fig. 7.1), hierarchically organized according to the abstraction level of the capability provided and the provider's service model. The three models are (1) Infrastructure as a Service (IaaS), (2) Platform as a Service (PaaS), and (3) Software as a Service (SaaS) (Mell and Grance 2011). The different providers that we witness today, have particularly developed competences related to these different technical layers (i.e., software, platform, and infrastructure) (Marston et al. 2011).

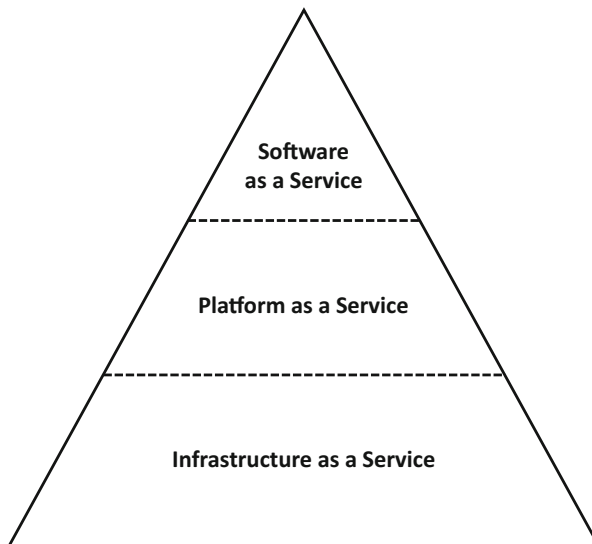


Fig. 7.1 Cloud service models depicted as a pyramid

Infrastructure as a Service (IaaS)

Cloud customers can procure processing, storage, networks, and other fundamental computing resources by using IaaS services. They can utilize these resources by deploying and running arbitrary software, which can include operating systems and applications (Mell and Grance 2011). The provider typically provides this service by dividing a very large physical infrastructure resource into smaller virtual resources for access by the customer (Arasaratnam 2011). Sometimes the cloud service the

provider offers is a complete virtual machine with an operating system. In other instances, the cloud service offers storage capacities or a bare virtual machine with no operating system. Cloud customers do not manage or control the underlying cloud infrastructure, but they have control of the operating systems, storage, and deployed applications. Also, customers' limited control of select networking components (e.g., host firewalls), is possible (Mell and Grance 2011). For example, *Amazon Web Services* mainly offers IaaS by providing virtual machines with a software stack that can be customized similarly to a conventional physical server (Voorsluys et al. 2011). Cloud customers are given privileges to perform various activities on the virtual machine, such as starting and stopping it, customizing it by installing software packages, and configuring access permissions and firewall rules.

Examples for IaaS include: backing up and recovering file systems and data stored on servers and desktop systems; delivering server resources for running cloud-based systems that can be dynamically provisioned and configured as needed; managing cloud infrastructure platforms; providing massively scalable storage capacity services that can be used for applications, backups, archival, and file storage (Liu et al. 2011). Leading cloud providers offering IaaS, among others, are *Amazon*, *IBM*, *Microsoft*, *Rackspace*, *NTT*, *Oracle*, and *Fujitsu* (ITCandor 2018).

Platform as a Service (PaaS)

Cloud customers can deploy customer-created or acquired applications to the cloud infrastructure by using programming languages, libraries, services, and tools supported by the cloud provider (Mell and Grance 2011). A cloud customer does not manage or control the underlying cloud infrastructure (e.g., the network, servers, operating systems, or storage), but can control the deployed applications and also the configuration settings for the application-hosting environment. Thereby, PaaS offers a highly integrated environment on which developers design, build, test, and deploy custom applications without the costs and complexity of buying and managing the underlying hardware and software layers (Schneider and Sunyaev 2016). *Google's App Engine* is an example of a PaaS that offers a scalable environment for developing and hosting web applications (Voorsluys et al. 2011). Its building blocks are, among others, an in-memory object cache, a mail service, an instant messaging service, an image manipulation service, and integration with Google's account authentication service.

Further examples of PaaS offerings, are platforms for application development and testing, services offering scalable relational or NoSQL databases, dedicated development platforms for building integration applications in the cloud and within the enterprise, and runtime environments suited to run specific applications (Liu et al. 2011). Leading providers offering PaaS, among others, are *Amazon*, *Microsoft*, *Alibaba*, *Google*, *IBM*, and *Rackspace* (Gartner 2018b).

Software as a Service (SaaS)

In this model, cloud customers use a cloud provider's applications running on a cloud infrastructure (Mell and Grance 2011). Applications are typically accessible from various client devices through either a thin client interface, such as a web browser, or a program interface. The customers do not manage or control the

underlying cloud infrastructure (e.g., the network, servers, operating systems, storage, or even individual application capabilities), with the possible exception of limited user-specific application configuration settings. Thus, a cloud customer, in most cases, is completely abstracted from the nuances of the application running behind the scenes (Arasaratnam 2011). Due to SaaS, customers are increasingly shifting from locally installed computer programs to online software services that offer the same functionality while also alleviating customers’ possible burden of software maintenance (Voorsluys et al. 2011).

Examples of SaaS are: applications for email, word processing, spreadsheets, and presentations; application services to manage customer billing based on usage and subscriptions to products and services; customer relationship management applications that range from call center applications to sales force automation; tools that allow users to collaborate in workgroups within enterprises and across enterprises; applications for managing financial processes ranging from expense processing and invoicing to tax management. Leading cloud providers offering SaaS, among others, are *Salesforce*, *Microsoft*, *SAP*, *Oracle*, *Adobe Systems*, and *IBM* (Apps Run The World 2017). The company Loudcloud is supposed to be one of the first vendors that talked about cloud computing and SaaS back in 1999.

Besides the fundamental classification of cloud service models into PaaS, IaaS, and SaaS, there are many other service models in practice and the literature. The latter are often summarized as Everything as a Service (XaaS) (Singh et al. 2016). Table 7.1 lists examples of such service models, including Database as a Service or Security as a Service, and shows how they relate to the fundamental service models (see also Höfer and Karagiannis (2011)).

Table 7.1 Cloud service models and their assignment to the fundamental service models, software, platform and infrastructure as services

Service model	Fundamental service models			Exemplary Literature
	SaaS	PaaS	IaaS	
Security as a service	●	-	-	Sharma et al. (2016)
Search as a service	●	-	-	Dašić et al. (2016)
Testing as a service	-	●	-	Linthicum (2009)
Database as a service	-	●	-	Linthicum (2009)
Network as a service	-	-	●	Soares et al. (2011)
Rendering as a service	-	-	●	Annette et al. (2015)

7.1.5 Cloud Computing Deployment Models

The NIST definition of cloud computing distinguishes between four basic deployment models: private, public, community, and hybrid cloud (Lins et al. 2019a; Mell and Grance 2011). In addition, virtual private and multi-cloud deployment models are often discussed in the literature and practice.

Private Cloud Models

The private cloud infrastructure is used only by a single person or organization and its members (Mell and Grance 2011). It can be owned, managed, and operated by the person or organization, by a third party, or by a combination of both. The private cloud generally serves internal company purposes, and cloud customers have full control over who, how, and when a cloud service can be used. For example, a private cloud service could be utilized by a financial company that seeks to store sensitive data, yet still wants to benefit from the advantages of cloud computing, such as on-demand resource allocation. Multiple cloud providers – including *Amazon*, *IBM*, *Cisco*, *Dell* and *Red Hat* – build private solutions.

Public Cloud Models

The cloud infrastructure can also be used by the general public (Mell and Grance 2011). Companies, academic or government organizations, or a combination of these, own, manage, and operate the cloud infrastructure. A public cloud generally provides a selection of services simultaneously for all users (in the form of, e.g., business processes, business practices, applications, and infrastructures). Such services are provided on the basis of usage-dependent payment via the Internet. One of the most significant distinguishing features of a public cloud is that a cloud customer can neither technically, nor contractually, influence which other parties use the cloud service. Thus, cloud customers (unknowingly, in terms of extent and scope) share the underlying infrastructure, which is, however, completely abstracted from the application layer. Examples of public clouds include *ESDS's eNlight Cloud*, *Amazon Elastic Compute Cloud (EC2)*, *IBM's Blue Cloud*, *Sun Cloud*, *Google App Engine* and *Windows Azure Services Platform*.

Community Cloud Models

The cloud infrastructure is used exclusively by a group of people or organizations who have similar demands of the cloud service, such as sharing the same mission, security requirements, policy, or compliance considerations (Mell and Grance 2011). One or more of the community members, third parties, or a combination of these parties own, manage, and operate the cloud infrastructure.

For example, IBM offers its *Federal Community Cloud* to federal government organizations as part of the company's dedicated federal data centers service, which provides secure certified computing capabilities. Cloud customers can decide on community solutions that *Google*, *Red Hat*, *IBM*, *Microsoft*, or others provide.

Hybrid Cloud Models

The hybrid cloud infrastructure consists of a combination of two, or more, of the above-mentioned models (in particular public and private cloud). The individual infrastructures remain unique entities, but are connected by standardized or proprietary technologies. This allows the transfer of data and applications between the connected infrastructures. The purpose of this mixed form of services is to create a solution that best meets the concrete requirements of each company. For example, a cloud customer could run a mission-critical workload within a private cloud, but use the database services of a public cloud provider for non-critical data.

Virtual Private Cloud Models

The term “virtual private cloud” was first used widely by *Amazon Web Services* when its new product, “*Amazon VPC*”, was introduced. In the virtual private cloud model, the cloud provider supplies the underlying infrastructure exclusively to a single organization, which could include a number of users (e.g., business divisions) (Dillon et al. 2010). Access to the cloud service can be realized using a Virtual Private Network. The cloud infrastructure remains the cloud provider’s property, and is thus operated and managed by the cloud provider. A virtual private cloud ensures that a cloud customer has complete control of the virtual resources.

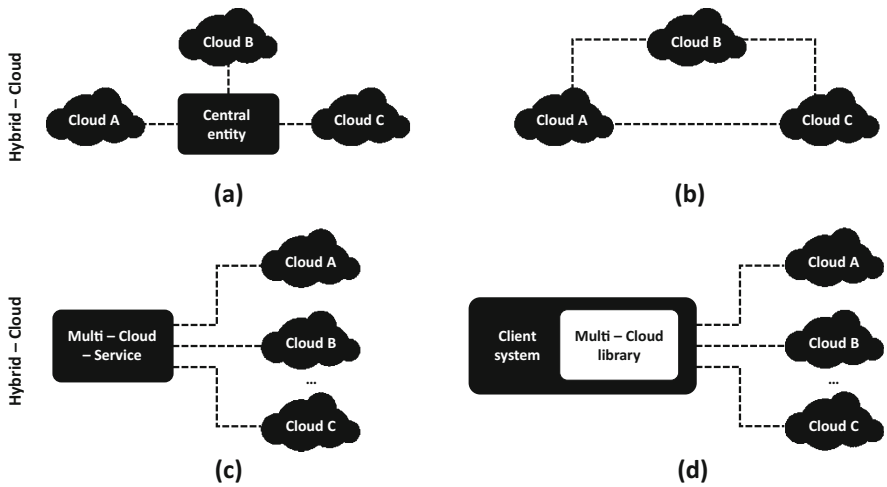


Fig. 7.2 Illustrative types of hybrid and multi-clouds (adapted from Grozev and Buyya (2014))

Multi-Cloud Models

If cloud services of different cloud providers are aggregated and combined, this is referred to as a multi-cloud (Grozev and Buyya 2014). Here, cloud providers can voluntarily connect their cloud infrastructures and services with other cloud providers, or a cloud broker enters the market, aggregating different cloud services from (different) cloud providers. The distinction between a multi-cloud and a hybrid cloud inconsistent and difficult to articulate. In contrast to a hybrid cloud, in which the cloud infrastructures are usually connected and work together (orchestrated), in multi-clouds, only certain cloud service components are specifically used by another cloud provider. For example, a multi-cloud provider could perform the computing and network operations in an *Amazon Web Service* cloud, while storage is performed solely by the *Microsoft Azure* cloud. Fig. 7.2 illustratively represents hybrid and multi-cloud scenarios.

7.1.6 Differences Between Related Concepts

IT Outsourcing vs. Cloud Computing

IT outsourcing refers to transferring some or all of the IT related decision-making rights, business processes, internal activities, and services to external providers (Böhm et al. 2011). Cloud computing is an evolution and a specific form of IT outsourcing, and thus shares common characteristics, providing similar benefits to customers (Schneider and Sunyaev 2016). For instance, customers perform IT outsourcing or adopt cloud services to reduce costs, to minimize risk by using third party vendors, and to outsource control and application management, as the third party carries out these processes. Customers share similar security and privacy risks in IT outsourcing and cloud computing contexts, as either way, the data is handled by a (potentially untrustworthy) third party. From a consumer's perspective, the main distinguishing characteristic of cloud computing compared to traditional IT outsourcing, is the flexible deployment of virtual and asset-free resources and services (Böhm et al. 2011). This allows the implementation of flexible, pay-per-use business models. Cloud providers are able to provide existing customers with a new kind of flexibility, and they can reach entirely new customer groups by offering new services and business models. In addition, the cloud computing model allows for the modification and extension of existing services without large upfront investments. The cloud provider *JungleDisk*, for example, used the IaaS services of *Amazon* to build a new business model that offers end-users easy to use storage services.

Further peculiarities of cloud computing distinguish it from IT outsourcing (Schneider and Sunyaev 2016). For instance, compared to traditional IT outsourcing, cloud computing induces a shift in task responsibilities during decision processes and self-service procurement; provides standardized services with a narrower scope; enables new scenarios of outsourcing and governance arrangements; and, uses short-term usage-based contracts.

IT Outsourcing

IT outsourcing refers to transferring some or all of the IT related decision-making rights, business processes, internal activities, and services to external providers (Böhm et al. 2011).

Application Service Provision vs. Cloud Computing

Application service provision is a form of outsourcing in which organizations rent packaged software and associated services from a third party (Bennett and Timbrell 2000). In cloud computing, SaaS is the closest corresponding service model to application service provision. Researchers argue that SaaS emerged as an advanced form of application service provision (Benlian and Hess 2011), that cloud computing has the same key attributes as the more traditional application service provision model, and that it exposes users to the same risks (Schwarz et al. 2009). Further, application service provision and SaaS share similar business and pricing models (Weinhardt et al. 2009). However, early providers offering application services

failed due to insufficient bandwidth and computing power (Susarla et al. 2003). Finally, cloud computing offers also more diverse service models than providing applications as a service.

Application Service Provision

Application service provision is a form of outsourcing in which firms rent packaged software and associated services from a third party (Bennett and Timbrell 2000).

Grid Computing vs. Cloud Computing

Grid Computing enables resource sharing and coordinated problem solving in dynamic, multi-institutional, and large-scale geographically-distributed virtual organizations (Foster et al. 2001). Grids provide a distributed, peer-to-peer computing paradigm or infrastructure that spans multiple virtual organizations; each virtual organization consists of either physically distributed institutions or logically related projects/groups (Foster et al. 2008). The goal of such a paradigm is to enable federated resource sharing in dynamic, distributed environments. One famous example of grid computing is the SETI project. The SETI application looks for radio signals or other forms of communications in space. Given a high demand for computing resources, SETI developed a grid computing middleware where the application can be executed over multiple computers that are connected to the Internet. Individuals can then download and install the middleware, which used idle computing powers at the disposal for SETI.

Clouds and grids share a great deal in their vision, architecture, and technology (Foster et al. 2008). The evolution of cloud computing resulted from a shift in focus from an infrastructure that delivers storage and computing resources (as is the case in grids) to one that is economy-based and aims to deliver more abstract resources and services (as is the case in clouds). The vision of grids and clouds remains the same, namely to reduce the cost of computing, increase reliability, and increase flexibility by transforming desktop computers and internal IT infrastructures to a facility operated by a third party. Clouds and grids share functions, but also some common challenges, such as a need for managing large facilities, for defining methods by which customers discover, request, and use resources provided by the central facilities, and a need for implementing the often highly parallel computations that are executed on those resources. Nevertheless, clouds and grids also differ in various aspects such as security, programming model, business model, computing model, data model, applications, and abstractions (Foster et al. 2008). In a cloud-based business model, a customer usually pays the cloud provider on a consumption basis, very much like paying for electricity, gas, and water. The model relies on economies of scale in order to drive prices down for customers and profits up for providers. In contrast, the business model for grids is project-oriented, which involves that users or a community have a certain number of service units (i.e., CPU hours) they can spend to achieve the project objectives. Further, clouds mostly comprise dedicated data centers that belong to the same organization. Within each data center hardware

and software configurations, as well as supporting platforms, are generally more homogeneous. Grids, however, build on the assumption that resources are heterogeneous and dynamic, so that each grid site might have its own administration domain and operation autonomy.

Grid Computing

Grid Computing enables resource sharing and coordinated problem solving in dynamic, multi-institutional, and large-scale geographically-distributed virtual organizations (Foster et al. [2001](#)).

7.2 Essentials to the Provision of Cloud Services

7.2.1 Essential Cloud Technologies

Cloud services use a set of technology components that enable key features and characteristics of cloud computing (Singh and Chatterjee [2017](#)). This section defines such technologies that help with understanding how these technologies are used to build cloud infrastructures.

Broadband Network and Internet Technology

The Internet allows customers to access remote cloud resources in order to support ubiquitous network access (Singh and Chatterjee [2017](#)). Cloud computing's inherent network access requirement creates a built-in dependency on the Internet or private networks. The Internet is established and deployed by the Internet Service Provider (ISP). Each ISP can freely choose, manage, and add another ISP to their networks.

Data Center Technology

Data center technology contains multiple technologies and components that are typically connected to one another (Singh and Chatterjee [2017](#)). The data center is built with standardized commodity hardware and modular architecture, multiple aggregation, unique building blocks that provide scalable, incremental growth of the services and reduce cost of investment in the cloud. The data center has both physical and virtualized IT resources. Physical IT resources include networking systems, servers, and equipment. Virtualized IT resources are placed in the virtualization layer that is operated and managed by the virtualization platform. Data center automation is also becoming increasingly important because automating the bulk of the data center operations, management, monitoring and maintenance tasks that otherwise are performed manually by human operators increases efficiency and reduces costs.

Virtualization Technology

Virtualization refers to the conversion process that translates physical IT resources into virtual IT resources, such as virtual machines (Singh and Chatterjee [2017](#)). The virtual IT resources include servers, storage, network, and computing power. Virtualization conceals the physical characteristics of IT resources and instead, presents an abstract, emulated virtual machine (Marston et al. [2011](#)). A hypervisor

manages virtualized IT resources by running them on the physical host and providing coordination capacities (Bayramusta and Nasir 2016). The virtualized computing infrastructure is much better utilized than a purely physical infrastructure, which leads to lower upfront and operational costs. Virtual machines behave like an independent system, but unlike a physical system they can be started, closed down, and configured on demand; they can also be maintained and replicated very easily. Modern virtualization techniques allow cloud services to transfer virtual machines from a server with less space to another that has more space. This ability to re-allocate storage and computing power increases the flexibility and scalability of computing operations.

Container Technology

Containers refer to a similar but lighter virtualization concept (see Fig. 7.3). Containers are less resource and time-consuming, thus they have been suggested as a solution for improved interoperable application packaging in the cloud (Pahl 2015). They offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. Such detachment allows for easy and consistent deployment of container-based applications, regardless of whether the target environment is a private data center or a public cloud. Containerization provides a clean separation of concerns, as developers focus on their application logic, while IT operations teams focus on deployment and management of containers without worrying about the application details, such as versioning and specific configurations of the application. Although virtual machines and containers are both virtualization techniques, they solve different kinds of problems. Containers are tools for portably delivering software that aims at greater interoperability while still utilizing software virtualization principles. Virtual machines, on the other hand, take care of hardware allocation and management, with machines that can be turned on/off. Containers can replace virtual machines if the allocation of hardware resources is done using containers that divide workloads between clouds.

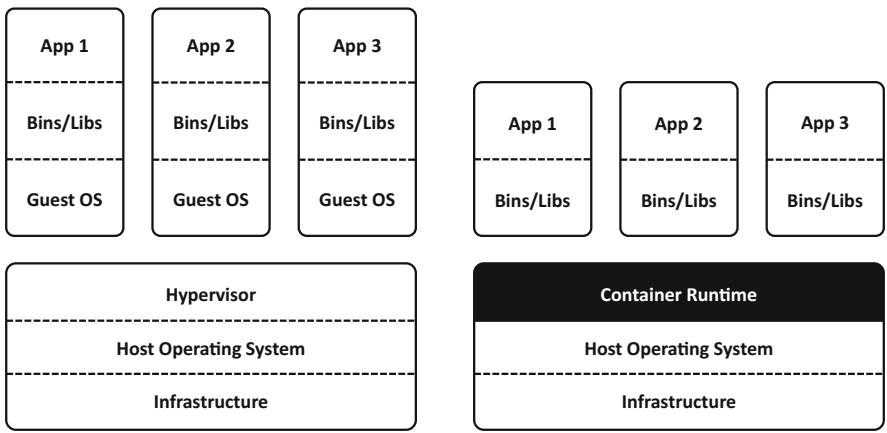


Fig. 7.3 Comparison of virtual machines and containers

Multi-Tenant Technology

Multi-tenant technology enables multiple users to access the same application logic simultaneously (Singh and Chatterjee 2017). Multi-tenant applications ensure that each tenant has a separate own view of the application, and tenants are not allowed to access data and applications provided to other tenants. Tenants can individually manage their features of the application. Such features are the user interface, business process, data model, and access control. The most important challenges multi-tenant applications face relate to usage isolation, data recovery, data tier isolation, data security, application upgrades, system scalability, and metered usage.

Service Technology

Service technology is the basic foundation of cloud computing, used for creating “as-a-service” cloud delivery models (Singh and Chatterjee 2017). Web service, REST service, service agents, and service middleware are basic technologies for building the cloud-based environment.

7.2.2 Cloud Service Stack

Cloud computing is based on a network of interrelated services on different levels of abstraction that form a layered architecture, which is also referred to as the ‘cloud stack’. This layered architecture mirrors the various cloud computing service models (i.e., IaaS, PaaS, SaaS layers). The cloud stack helps to delineate responsibilities between cloud provider, cloud customers, as well as sub-providers. Table 7.2 schematically outlines the potential responsibilities. Depending on the cloud services’ architecture, deviations to this simplified representation could occur in practice. Moreover, a cloud service is not limited to a specific operational environment or layer of the cloud stack. The layers can be spread out across all levels and, in the sense of the networked service structure, can be operated simultaneously by legally independent sub-providers. Thus, different parties and a combination of parties (e.g., cloud providers and sub-providers) can be responsible for platform security.

The following paragraphs will exemplify the separation of responsibilities across the three main cloud service models SaaS, PaaS, and IaaS. When using a SaaS model, a cloud customer generally is unable to make any changes to the cloud service. A cloud customer is only able to adjust certain configurations or settings on related cloud applications, such as turning certain features on or off, or customizing graphical user interfaces. Notably, the cloud customer is responsible for using the cloud application securely and in compliance with existing regulations. The SaaS provider’s main responsibilities include the development, operation, maintenance, and administration of the software application, as well as ensuring software security. The remaining cloud layers can be operated by the SaaS cloud provider (full-stack provider) or outsourced to a sub-provider.

Table 7.2 Illustration of the different layers of the cloud service stack

Party	IaaS	PaaS	SaaS	Description of the layer	
Cloud customer	Secure application usage			The cloud customers are responsible for secure usage of the application.	
	User specifics			User-specific settings or configurations of used applications.	
	Application			Offered software solutions.	
	Software security			Mechanisms to increase the security of provided applications.	
	Administration and support			Administration of the software as well as receipt and handling of support inquiries from the cloud customer.	
	Operating system	Operating system	Operating system	Basic software for operation of the application.	
	Runtime environment	Runtime environment	Runtime environment	The runtime environment carries out applications for which the runtime environment is suitable.	
	Database	Database	Database	Software for the management and structuring of data.	
	Platform security	Platform security	Platform security	Mechanisms to increase the security of provided platforms.	
	Administration and support	Administration and support	Administration and support	Administration of the provided platform as well as receipt and handling of support inquiries from the cloud customer.	
Virtual machines	Virtual machines			Virtual representation of computer resources like, for example, servers or CPUs.	
Cloud provider	Virtualization	Virtualization			Mechanisms for the creation and management of virtual machines.
	Computing Storage	Computing Storage			Components to process data in the cloud service.
	Network	Network			Mechanisms for the storage of data.
	Infrastructure security	Infrastructure security			Mechanisms for the transfer of data.
	Administration and support	Administration and support			Mechanisms to increase the security of provided resources.
	Administration and support			Administration of the provided infrastructure as well as receipt and handling of support inquiries from the cloud customer.	
	Hardware			The physical hardware for operation of the cloud service.	
	Premises, housing and equipment			The physical set-up of the cloud service.	
	Network			Physical connectivity of the data center.	
		Data center security			Mechanisms to increase the security of the data center, including parties responsible for the housing, with security staff and physical security systems.

Key:

Cloud customer

IaaS core business

SaaS core business

PaaS core business

IaaS core business

Full stack provider or sub-provider

In the case of a PaaS, cloud customers deploy and run their own applications on a cloud platform. Thus, a cloud customer is responsible for the development and operation of the applications, and has the possibility of some control over the hosting environment settings. Additionally, a cloud customer is responsible for securing the application, for example, to prevent cross-site scripting or software flaws. The PaaS provider's main responsibilities are the development, operation, and administration of the platform, caring for components such as runtime software execution stack, databases, and other middleware components, as well as ensuring platform security. PaaS cloud providers typically also support the development, deployment, and management of the cloud customer by providing tools, such as integrated development environments (IDEs), software development kits, deployment, and management tools (Liu et al. 2011). The remaining cloud layers can be operated by the PaaS cloud provider (full-stack provider) or are outsourced to a sub-provider.

The cloud provider of the IaaS is responsible for securely virtualizing and providing the necessary physical resources. The IaaS cloud provider typically has control over the hardware and cloud software that makes the provision of these infrastructure services possible. For example, it oversees the physical servers, network equipment, storage devices, host operation systems, and hypervisors for virtualization (Liu et al. 2011). A cloud customer is responsible for provisioned virtual machines and the applications, databases, operating systems, and runtime environments. In addition, the cloud customer takes responsibility for software and platform security. The required physical hardware, housing, and equipment can be acquired and managed by an IaaS provider (full-stack provider) or can be obtained from a sub-provider.

Depending on the specific design of the cloud stack and the service level agreements made with cloud customers, different responsibilities could arise. Regarding further literature, readers could refer to the *NIST 'Cloud Security Reference Architecture'*, which provides a detailed exposition of sharing responsibilities in various service models (NIST Cloud Computing Security Working Group 2013; Liu et al. 2011).

7.3 Chances and Challenges of Cloud Computing

7.3.1 *Reasons to Move into the Cloud: Benefits and Opportunities for Organizations*

Essentially, cloud computing enables individuals and organizations to access IT resources on-demand, from any device, at any time, and as a measured service. Due to its inherent characteristics (e.g., on-demand self-service, resource pooling, elasticity, and extensibility), cloud computing enables persons and organizations to achieve diverse benefits and opportunities. More importantly, cloud computing

exhibits transformative mechanisms that enable truly innovative services and business models, that ultimately lead to fundamental and large-scale innovations that benefit individuals, organizations, markets, and societies (Benlian et al. 2018). In the following sections, major benefits for organizations moving to the cloud will be presented, as well as the transformative mechanisms of cloud computing.

Low Entry Barriers

One of the cited benefits of cloud computing is a lowered entry barrier to large data centers that provide unique services for niche markets previously inaccessible due to high capital costs (Arasaratnam 2011; Venters and Whitley 2012). This is facilitated by customers sharing a significant amount of infrastructure which reduces costs per customer. This allows for economies of scale from both a service management and computing resource perspective. Cloud computing, therefore, enables small and medium-sized organizations to enter new markets quickly, particularly in areas like business intelligence which previously required significant IT investment (Venters and Whitley 2012). For emerging countries, cloud computing offers yet another opportunity to ‘leapfrog’ to advanced technology by connecting to cloud providers outside their countries (Venters and Whitley 2012). Some cloud computing providers are using the advantages of a cloud platform to enable IT services in countries that would traditionally have lacked the resources for widespread deployment of IT services (Marston et al. 2011).

Pay-as-you-go

Cloud services have brought about the reclassification of IT from an expensive ‘capital expenditure’ (requiring large upfront investment which could be difficult to raise) to a pay-as-you-go ‘operating expenditure’ (Venters and Whitley 2012). For example, cloud services enable universities and education institutions to provide computing laboratories to students and bulk processing services for research on a pay-per-use basis (Sarkar and Young 2011; Venters and Whitley 2012). Customers can thereby optimize the use of IT resources by transferring fixed costs to variable cost (Baldwin et al. 2001). This approach affects the customers’ cash flow, with small, stable cash flows occurring rather than large periodic payments for licenses, maintenance contracts, and upgrades (Benlian and Hess 2011).

Access to Leading Edge IT Resources, Skills, and Capabilities

Access to leading edge IT resources has been shown to be one of the main indicators of IT outsourcing success, and an important driver of outsourcing decisions (Gonzalez et al. 2009; Benlian and Hess 2011). Cloud providers benefit from economies of scale by using a multi-tenant platform architecture, and consolidating and virtualizing its data centers. Further, the provider also benefits from learning curve effects when the delivery of cloud services via the Internet is professionalized (Benlian and Hess 2011). As a result, cloud customers benefit from economies of skill by leveraging the skills, resources, and capabilities that the service provider offers. These specialized capabilities (e.g., access to the latest technologies and IT-related know-how) could not be generated internally if the IT resources were delivered in-house via an on-premises model (Kern et al. 2002). Other business

benefits of cloud can include a reduced demand for skilled labor where skills shortages exist (Luftman and Zadeh 2011).

Quality Improvements

Customers frequently rely on cloud services to improve the quality and productivity of IT services by outsourcing to a third-party vendor (Benlian and Hess 2011; Schneider and Sunyaev 2016). They expect providers to incorporate industry's best practices and total quality management procedures, such as lean management concepts (Benlian and Hess 2011), and to aim for various quality improvements, such as a faster response time to end-users, or higher-quality user interfaces and features (Bajaj 2000). In general, cloud services are assumed to provide better reliability and availability due to a robust architecture. For example, when Netflix migrated to the cloud, not only the service quality was improved, but Netflix product itself has continued to evolve rapidly, incorporating many new resource-hungry features and relying on ever-growing volumes of data. Cloud-based business relationships between providers and customers, which are commonly based on key performance indicators (such as end-user response time or net uptime), allow for increased (outcome) measurability of service quality. Such relationships also enable clear contractual specifications regarding adequate service levels (e.g., with explicit provisions, which include fines, penalties, and even contract termination) (Benlian and Hess 2011). These characteristics in turn allow for higher transparency than that of an internal IT department, and could translate into stronger cloud provider discipline and better service quality.

Cost Savings

Cloud customers seek to achieve cost advantages by relying on cloud services, because external cloud providers can provide IT functions, such as managed application services, at lower costs than customers can (Benlian and Hess 2011; Venters and Whitley 2012). This ability is due to the providers' specialization and achievement of economies of scale and scope. Since the cloud business model runs a shared infrastructure and provides multiple users with a single instance of a highly standardized software service, it can be based on an extremely scalable and cost-efficient platform. Such improved economics in the provision of cloud-based resources can be passed on to customers, who can benefit from the lower total ownership costs. Treating IT as an operational expense helps to reduce the upfront costs in corporate computing quite dramatically. For example, many of the promising new Internet startups like *37 Signals*, *Jungle Disk*, *Gigavox*, and *SmugMug*, were realized with investments in information technology that are orders of magnitude lesser than those that would have been required just a few years ago.

Focus on Core Capabilities

In general, outsourcing enables organizations to focus on their core business, because they can free up resources, which can be used more productively in areas that create value (Benlian and Hess 2011; Schneider and Sunyaev 2016). This refocusing is possible by completely shifting the responsibility for developing,

testing, and maintaining the outsourced resources and the underlying infrastructure to the cloud provider. This shift will not only relieve line managers of the task of coordinating a large IS department, but will also eliminate IT staff members' routine support activities (Gonzalez et al. 2009). The staff can then dedicate their time to more strategic activities to determine how the organizations' IT can add value to the business.

Greater Flexibility and Elasticity

Using cloud services provides a great degree of flexibility regarding the utilization of easily scalable and on-demand provision of IT resources (Benlian and Hess 2011). Customers can quickly scale resources both up and down and pay as you go based on the amount customers actually use. This flexibility makes it easier for organizations to respond to business-level volatility, because the cloud provider handles fluctuations in IT workloads. In this regard, a customer can leverage a cloud provider's capacity to adapt to change. Since the computing resources are managed through software, they can be deployed very quickly as new requirements arise (Marston et al. 2011).

Reduced Time to Market

When services are sourced externally, cloud computing enables customers to deliver their products or services to the market faster (Seethamraju 2013). Cloud providers afford almost immediate access to hardware resources, with no upfront capital investments for users, which leads to a faster time to market in many businesses (Marston et al. 2011).

Lower IT Barriers to Innovation

Cloud computing can lower IT barriers to innovation, as the many promising startups testify, like *TripIt* (for managing travel arrangements) or *Mint* (for managing personal finances) (Marston et al. 2011). Cloud computing also makes possible new classes of applications, and delivers services that were not possible before. Examples include: (1) interactive mobile applications that are location, environment, and context-aware and that respond in real time to information provided by human users, nonhuman sensors (e.g., humidity and stress sensors within a shipping container), or even by independent information services (e.g., worldwide weather data); (2) Parallel batch processing, that allows users to take advantage of huge volumes of processing power to analyze terabytes of data for relatively small periods of time, while programming abstractions like *Google's MapReduce* or its open-source counterpart *Hadoop* make the complex process of parallel execution of an application over hundreds of servers transparent to programmers; (3) Business analytics that can use the vast body of computer resources to understand customers, buying habits, supply chains, and so on, from voluminous amounts of data; (4) Extensions of compute-intensive desktop applications that can offload the data crunching to the cloud, leaving only the rendering of the processed data at the front-end, with the availability of network bandwidth reducing the latency involved.

7.3.2 *Cloud Computing's Transformative Mechanisms*

Whereas cloud computing has traditionally been viewed as a means to generate IT value, e.g., economic advantages through the deployment and use of IT (Schneider and Sunyaev 2016), cloud computing exhibits transformative mechanisms that enable truly innovative services and business models, ultimately leading to fundamental and large-scale innovations that benefit individuals, organizations, markets, and societies (Benlian et al. 2018). In the following sections, we describe three transformative mechanisms of cloud computing, namely decoupling, platformization, and recombination.

Decoupling

Decoupling describes a process in which one element of a system becomes an independent service with a defined service interface so that internal changes in this service will not disrupt the functioning of other dependent elements (Tiwana et al. 2010). In cloud computing, decoupling started on the infrastructure level (Benlian et al. 2018). Enabled by virtualization techniques, application systems have become independent of their underlying physical resources. For example, the detachment of large resource-intensive applications run on virtualized layers that utilize multiple physical instances. Virtualization allows for elastically scaling these resources up and down—a key characteristic of cloud computing. Cloud computing also has exploited and spurred on the increasing decoupling on the component level. While modularization and decoupling has long been a trend in software engineering in both monolithic and distributed systems, widespread web service protocols (e.g., SOAP, REST) have increased the level of decoupling and made the reuse of third-party services a more common practice. Applications use functionality and resources from remote services, e.g., for retrieving geolocation information from *Google Maps*. On the cloud application level, decoupling has led to a greater separation between those who use applications and those who provide them—a logical continuation of the outsourcing trend (Schneider et al. 2018; Schneider and Sunyaev 2016). Traditional IT outsourcing arrangements were single-tenant with IT resources dedicated to a specific user organization (Yoo et al. 2010). Cloud computing services are provided for use by multiple tenants with a postulate of minimal service provider interaction, which further decouples the provider-user relationship.

Decoupling

Decoupling describes a process in which one element of a system becomes an independent service with a defined service interface so that internal changes in this service will not disrupt the functioning of other dependent elements (Tiwana et al. 2010).

Platformization

The second mechanism, platformization, builds on decoupling and characterizes the process in which an entity (a provider organization) creates access and interaction opportunities centered around a core bundle of services (the platform) within an ecosystem of customers, complementors (i.e., cloud service partners), and other stakeholders (Tiwana et al. 2010; Cusumano 2010). On all cloud computing layers, new players have emerged that develop capabilities by providing infrastructure resources, applications, and component services, leading to the commercial reality of IaaS, SaaS, and PaaS (Benlian et al. 2018). Customers harness cloud platforms to gain access to easily connectable services, and are able to satisfy their unique needs due to cloud platforms providing greater variety, service quality and business flexibility than otherwise. On the resource layer, *Amazon*, for example, made unused virtualized computing resources available to the external market with offerings like the *EC2* (the elastic cloud) and swiftly became a market leader for IaaS. From a customer perspective, the large data centers of public cloud providers enabled practically infinite possibilities for scaling computing resources. On the component level, PaaS providers, such as *Google App Engine*, *IBM Bluemix*, and *Heroku*, bundled online development and execution environments with an increasing number of services, offering developers who design cloud-based applications from reusable components rather than building software from scratch. On the application level, cloud-native software vendors like *Salesforce.com* specialized in specific enterprise software segments, and thereby earned tremendous success since they provided cost-effective solutions to markets that would otherwise not have been able to afford these systems. These IaaS, PaaS, and SaaS providers have quickly turned into platform businesses and built up an ecosystem of customers, complementors (i.e., cloud service partners), and third parties that exchange knowledge and other resources with or via the platform (Grover and Kohli 2012). Cloud platform players have gained the critical size required for their ecosystems to flourish and to co-create new services, which ultimately benefit cloud customers.

Platformization

Platformization characterizes the process by which an entity (a provider organization) creates access and interaction opportunities centered around a core bundle of services (the platform) within an ecosystem of customers, complementors (i.e., cloud service partners), and other stakeholders (Tiwana et al. 2010; Cusumano 2010).

Recombination

The third transformative mechanism, recombination, refers to the process in which innovation potential is generated by combining cloud services and integrating them with other transformative technologies within and across platform ecosystems (Benlian et al. 2018). Recombination takes place on all three service models. For

example, IaaS platforms allow third-parties (complementors) to offer basic resources (e.g., pre-configured virtual instances, Internet of Things utilities) on marketplaces where users can choose from myriads of configuration options that will fulfill their specific needs. On the component level, PaaS has opened the door to the application programming interface (API) economy (Tan et al. 2016). Not only do PaaS platforms like *Amazon AWS*, *Microsoft Azure*, and *IBM Bluemix* allow complementors to offer value adding components (e.g., analytics, artificial intelligence, and blockchains); developers also integrate cross-platform via APIs with other open services from the programmable web to provide best-of-breed solutions that were formerly unthinkable. On the application level, all major software vendors aim to cultivate marketplaces with apps from third-party developers that offer similar innovation opportunities that the platform alone could never seize. *SAP App Center*, and *Microsoft Dynamics Marketplace* are only two examples from the large body of enterprise software vendors that have successfully created the app marketplace in the enterprise software world.

Recombination

Recombination refers to the process in which innovation potential is generated by combining cloud services and integrating them with other transformative technologies within and across platform ecosystems (Benlian et al. 2018).

7.3.3 The Downside of Cloud Computing: New Risks and Challenges

Although there are many benefits to cloud computing, it is not without its own risks and challenges. Cloud computing differs from previously studied products and services in the sense that it introduces ongoing uncertainty in the relationship between the provider and the customer (Trenz et al. 2018). Although customers depend on the cloud provider at all times, they typically lack personal interaction with the provider and have only limited information about the providers' qualities, intentions, and (future) actions. The sections below present several major risks and challenges concerning the use of cloud services. Specifically, the following subchapter discusses security and data protection issues in more detail.

Multitenancy

Although multitenancy affords cloud customers unprecedentedly low prices, security, privacy and compliance considerations need to be taken into account (Arasaratnam 2011). Depending on the layer of cloud service being provided, appropriate security controls should be employed. This can include host intrusion detection systems, hypervisor based security agents, host firewalls, and many others. Some cloud providers will enforce particular controls on their customers to ensure a minimal level of security. Others will allow their clients full flexibility. Cloud

customers who have concerns regarding the security of their workload in a multitenant environment should consult their provider regarding their security standards. If the provider's standards are deemed insufficient, the customer should address this with compensating controls, or defer to an alternative provider.

Loss of Control

Traditional computing models have permitted a high degree of control over compute resources (Arasaratnam 2011). Cloud computing, by virtue of abstraction, prevents the customer from having the same level of influence over the computing resource. In this regard, the cloud customer loses the administrative power, as well as operational and security control over the cloud system. In addition, cloud providers can outsource or sub-contract services to third-parties (i.e., sub-providers) that might not offer the same commitment as the provider does. Customers can only assess the provider's business operations and capability to fulfill transactions through visiting their website and consuming the provided online service itself. Hidden actions of the provider might never be detected because the customer cannot continuously monitor providers' actions. For instance, cloud customers can never fully assess whether cloud providers disclose customer's information to third parties without their consent (Trenz et al. 2018). In other cases, there can be a significant time lag before customers recognize providers' concealed actions, e.g., through media disclosure.

Location Intransparency

Location intransparency is a concern linked to social or regulatory contexts rather than technological issues (Ahmed and Litchfield 2018). If data from one region (e.g., nation and jurisdiction) is transferred to any other region, there is no guarantee that the data is being treated in the same way as the source would do. This includes the level of security, as well as retention and processing of data. Ideally, regulations that address data management should be consistent across jurisdictions. However, different countries and regions have different requirements regarding how its citizens' information should be handled (Arasaratnam 2011). In some areas, such as the European Union (EU), there are specific requirements regarding EU residents' data protection. In other areas, such as the United States, there are directives regarding protected health information, such as the Health Insurance Portability and Accountability Act. The challenge is that if the customer doesn't know where all the cloud provider's assets reside, it is difficult to determine with which legislation the customer has to comply. Further, if the cloud provider has multiple data centers worldwide, in many instances it is impossible to tell where in the world a particular customer's data set might be at any one point in time.

Lack of Availability

Customers have certain expectations about the service level to be provided once their applications are moved to the cloud (Voorsluys et al. 2011). These expectations include availability of the service, its overall performance, and what measures are to be taken when something goes wrong in the system or with its components. Typically, these expectations are settled in the SLA with the cloud provider. Yet,

cloud services face several threats regarding their availability. A nefarious user within the cloud can adversely affect the performance of or availability to other customers by attempting to over-consume resources (Arasaratnam 2011). A similar effect can result from customers using the significant elasticity of the cloud to launch a denial of service attack against other customers or organizations. Although most cloud providers have defense mechanisms against these attacks, law enforcement can be just as disruptive. Consider a recent case where law enforcement officials confiscated racks of servers from a data center due to alleged illegal activity by one of the customers. Unfortunately, due to the use of virtualization many tenants' information was confiscated at the same time. Hardware availability is another issue in cloud computing (Singh and Chatterjee 2017). If hardware resources are unavailable, it can lead to cloud outages that hamper the entire online business fraternity, thus causing considerable distress. For example, in late 2007, the cloud provider *Rackspace* suffered a 36-hour outage due to a damaged transformer outside their data center (Arasaratnam 2011). This significant issue affected all of their customers.

Compromised Ease of Use

Customers might be uncertain about the effort required to use the cloud service (Venkatesh et al. 2003). If the cloud service's computer interface is simple, easy to use, and does not require specialized skills, it is more likely to be perceived as useful (Lee and Wan 2010). Conversely, if the interface is difficult to understand and use, and requires specialized skills, it is likely to be perceived as less useful, in which case customers will be reluctant to adopt it. The degree of effort can be taken as a cue signaling that the system has been well tested and has the ability to work effectively. Such effort could also indicate that the provider is actually investing in the customer relationship (Gefen et al. 2003). Cloud services might exhibit a high degree of complexity, e.g., due to a broad range of functionalities, complex interface designs, or innovative capabilities.

Vendor Lock-In

Customers are confronted with uncertainty about whether they can leave the cloud service without incurring social or economic losses (Bhattacharjee and Park 2014; Trenz et al. 2013). Such losses are defined as switching costs, referring to one-off costs that customers associate with the process of switching from one provider to another (Burnham et al. 2003). There are different types of procedural switching cost, which are particularly relevant due to the low investment necessary to use a cloud service. Procedural switching costs comprise learning costs that arise from the time and effort of acquiring new skills or know-how required to use a new service effectively. This type of switching cost includes setup costs that represent the costs associated with the time and effort necessary for initiating a relationship with a new provider or setting up a new service. Further, switching costs for cloud services can be of a social nature (Jones et al. 2002), as is, e.g., reflected in the lost benefit of sharing files with other customers who use the same service. Likewise, a major

concern customers have is about having their data locked-in by a certain provider (Voorsluys et al. 2011). Users might want to move data and applications away from a provider that does not meet their requirements. However, in their current form, cloud computing infrastructures and platforms do not employ standard methods of storing user data and applications, and thus data portability is limited.

Limited Service Continuity

Another legitimate concern is centered on cloud providers: customers are uncertain whether the service (or particular service functions) will continue to be offered over time (Marston et al. 2011; ENISA 2012). As in any modern IT market, negative consequences resulting from, e.g., competitive pressure, an inadequate business strategy, provider acquisition, or lack of financial support, could put some providers out of business or at least force them to restructure their service portfolio offering. Subsequently, customers fear that in the short or medium term some widely used service functions, or even the complete cloud service, could be terminated. The impact of such a threat on the cloud customer is very understandable, because it could lead to a loss or deterioration of their performance in quality service delivery, as well as losses in financial and non-financial investment (i.e., learning costs).

7.4 Security and Data Protection in Cloud Environments

7.4.1 *Security and Privacy Challenges Due to Essential Cloud Service Characteristics*

Cloud services are an attractive alternative to traditional information technology for organizations in diverse industries (e.g., healthcare) due to cloud computing's essential characteristics (Gao et al. 2018; Thiebes et al. 2017). However, these features challenge contemporary approaches to security and privacy risk assessment (Benlian et al. 2018; Hentschel et al. 2018). For example, a multi-tenant and virtualized approach seems promising from a cloud provider's perspective in terms of profit, but this approach increases the risk of co-location attacks due to inappropriate logical and virtual isolation. Consequently, an increasing number of research and industry reports has focused on identifying and addressing security and privacy risks, including Internet, network, and access security issues, as well as risks regarding non-compliance with regulatory requirements (cf. Fernandes et al. (2014) for an excellent review of security issues in cloud environments). In the following sections, major challenges regarding the security and privacy of cloud services will briefly be presented.

Communication Issues

Communication with the cloud is performed via the Internet, which gives rise to many security risks during data transmission (Sehgal et al. 2011). Threats include

man-in-the-middle attacks, hardware-based attacks, browser and network vulnerabilities, as well as exploiting injection and protocol vulnerabilities. For example, man-in-the-middle attacks are performed by an intruder, who has access to packets moving in a network (Kouatli 2014). The intruder can interfere in communication between the cloud provider and customer, and would then be able to monitor and manipulate the traffic between them.

Data Breaches

A data breach is an incident in which sensitive, protected or confidential information is released, viewed, stolen, or used by an individual who is not authorized to do so (Top Threats Working Group 2016). The potential for data breaches increases exponentially in cloud service contexts, because cloud data centers comprise data from multiple customers (Whitman and Mattord 2011). A data breach can be the primary objective of a targeted attack, e.g., when organized crime seeks financial, health, and personal information to carry out a range of fraudulent activities. Likewise, competitors and foreign nations could be interested in proprietary information, intellectual property, and trade secrets. Activists might want to expose information that can cause damage or embarrassment. Data breaches could simply be the result of human error, application vulnerabilities, or poor security practices. Unauthorized insiders obtaining data within the cloud are a major concern for organizations.

Data Loss

Although customers outsource their data to the cloud because the cloud infrastructure is much more powerful and reliable, and has more capacity than their own devices or servers, customers still fear that their data can get lost (Burda and Teuteberg 2014; Park et al. 2016). From time to time outages and data loss incidents do occur in noteworthy cloud storage services (e.g., data loss incidents of storage providers *Linkup*, *Carbonite Inc.*, *Amazon's EC2* and *Google's Cloud*). Cloud services face a broad range of both internal and external threats to data integrity that can lead to data loss. Such threats include hardware or system malfunctions, human error, software corruption, or back-up failure (Subashini and Kavitha 2011). Similarly, data stored in the cloud can be lost due to the cloud provider accidentally deleting it, or worse, a physical catastrophe such as a fire or earthquake causing the permanent loss of customer data. Consequently, cloud providers typically operate multiple data centers to back data up in different locations.

Data Scavenging

Cloud servers typically reuse the storage space once the data has been properly utilized and sent to trash folders (Singh and Chatterjee 2017). In multi-tenant cloud environments, providers have to ensure that data used by a previous user is not available to the next user. In what is referred to as data scavenging, attackers could be able to recover removed data that, not having been destroyed properly, might still exist on the device, (Khan and Al-Yasiri 2016). The process of cleaning up or removing certain data units from a resource is known as data sanitization (Singh

and Chatterjee 2017). In distributed systems, data sanitization is a critical task in selecting the data destined for the trash, and for properly disposing of discarded data.

Insufficient Identity, Credentials, and Access Management

Cloud customers are often concerned about who can access their data in the cloud. Data breaches and successful attacks can occur due to lacking scalable identity access management systems, failure to use multifactor authentication, weak passwords, and a lack of ongoing automated rotation of cryptographic keys, passwords, and certificates (Top Threats Working Group 2016). In cloud service environments, implemented identity systems must scale to handle lifecycle management for millions of users. Consequently, identity management systems have to support immediate de-provisioning of access to resources when personnel changes, such as job termination or role change, occur. Multifactor authentication systems – smartcard or phone authentication, e.g. – are required for users and operators of a cloud service.

Cloud customers have to be aware that not only outsiders can gain access to their data, but that the provider itself is often able to access such data. Furthermore, in these cases the provider usually tries to conceal employees' access rights from the customers. In what is referred to as a malicious insider threat to an organization, the offender is a current or former employee, contractor, or other business partner who has or had authorized access to an organization's network, system, or data, and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information or information systems (Top Threats Working Group 2016). This threat mainly materializes due to lacking transparency and the IT services and customers working in a single management domain (Singh and Chatterjee 2017). An employee can gain a higher level of access, and using this, can penetrate and compromise the confidentiality of data and services. This can also result in an insider attacker attaining access to confidential data and affecting the cloud services' operation.

Insecure Interfaces and APIs

Cloud computing providers display a set of software user interfaces or APIs that customers use to manage and interact with cloud services (Top Threats Working Group 2016; Singh and Chatterjee 2017). The security and availability of general cloud services is dependent on the security of these basic APIs. From authentication and access control to encryption and activity monitoring, these interfaces must be designed to protect against both accidental and malicious attempts to circumvent policy. Further, organizations and third parties can build on these interfaces to offer value-added services to their customers. This introduces the complexity of the new layered API, and it also increases risk, because organizations could be required to relinquish their credentials to third parties in order to enable their agency. APIs and user interfaces are generally the most exposed part of a system, perhaps due to being

the only asset with an IP address available outside the trusted organizational boundary. Such assets will be acutely targeted for heavy attack. Thus, adequate controls protecting assets against Internet fraud are the first line of defense and detection. A successful attack on the cloud interfaces can result in root level access to a machine without initiating a direct attack on the cloud infrastructure (Hussain et al. 2017).

System Vulnerabilities

System vulnerabilities are exploitable bugs in programs that attackers can use to infiltrate a computer system with the purpose of stealing data, taking control of the system, or disrupting service operations (Top Threats Working Group 2016). Vulnerabilities within the components of the operating system (i.e. the kernel, system libraries, and application tools) put the security of all services and data at significant risk. This type of threat is nothing new, as bugs have been a problem ever since the invention of computers and thus became exploitable remotely when networks were created. With the advent of multitenancy in cloud computing, systems from various organizations are placed in close proximity to each other, and given access to shared memory and resources, creating a new surface for attacks.

Shared Technology Vulnerabilities

Cloud providers deliver their services in a scalable manner by sharing infrastructure, platforms, or applications (Top Threats Working Group 2016). Underlying components (e.g., CPU caches, GPUs, etc.) that comprise the infrastructure supporting cloud services deployment may not have been designed to offer strong isolation properties for a multitenant architecture (IaaS), re-deployable platforms (PaaS), or multi-customer applications (SaaS). This can lead to shared technology vulnerabilities that can potentially be exploited in all delivery models. A single vulnerability or misconfiguration can lead to a compromise across an entire provider's cloud. Flaws in hypervisors can allow one virtual machine to control or access information on another (Whitman and Mattord 2011).

7.4.2 Continuous Service Certification as Innovative Means to Ensure Security and Data Protection

A common strategy in reducing customers' security and data protection uncertainty that will also signal trustworthiness and adequate risk prevention, is the adoption of certification. This is particularly important for small- and medium-sized cloud providers because they cannot rely on a widely established high reputation in the market (Sunyaev and Schneider 2013; Lins et al. 2016a; Malluhi and Khan 2013). Certification is defined as a third-party attestation of products, processes, systems, or persons that verifies conformity to specified criteria (ISO 2004).

Certification

Certification is defined as a third-party attestation of products, processes, systems, or persons that verifies conformity to specified criteria (ISO 2004).

However, different existing cloud service certifications represent only a retrospective view of the fulfillment of technical and organizational measures when the relevant certificates are issued (Lins et al. 2016a; Lins et al. 2018). Typically, certification authorities evaluate adherence to certification criteria during a comprehensive audit performed once. Throughout the validity period of one to three years, certification deviations or breaches might not be detected until long after their occurrence because certification authorities validate certification adherence only via spot checks during annual surveillance audits. Thereafter, conventional certifications cannot support dynamic changes in the structure, deployment, or configuration of the cloud infrastructure that supports the provision of cloud services, such as the dynamic migration of data and software across different computational nodes in cloud federations (Krotsiani et al. 2015). In addition, a cloud provider can deliberately discontinue adherence to security and privacy requirements to achieve benefits, such as reducing required incident response staff to save costs (Lins et al. 2016a; Lins et al. 2018). Considering a highly dynamic cloud infrastructure, long validity periods of conventional certification could cause cloud customers to question the reliability and trustworthiness of issued certificates. This ultimately threatens the ability of cloud certification to prove adequate risk prevention (Lins et al. 2019b).

To address the juxtaposition of static certifications in an ever-changing and dynamic cloud service environment, researchers have started to investigate how the process of certifying cloud services can be innovated (Windhorst and Sunyaev 2013). These research efforts have resulted in innovative specifications of architectures and processes, as well as in continuous cloud service certification prototypes that allow certification authorities to continuously certify cloud services. Continuous cloud service certification (CSC) includes automated monitoring and auditing techniques, as well as mechanisms for transparent provision of certification-relevant information to continuously confirm adherence to certification criteria (Lins et al. 2016a). The process of continuous cloud service certification includes four major dimensions: (1) (semi-)automated data collection and transmission, (2) (semi-)automated data analysis, (3) up-to-date certification presentation, and (4) adjustment of processes (see Fig. 7.4).

Continuous Cloud Service Certification

Continuous cloud service certification includes automated monitoring and auditing techniques, as well as mechanisms for transparent provision of certification-relevant information to continuously confirm adherence to certification criteria (Lins et al. 2016a).

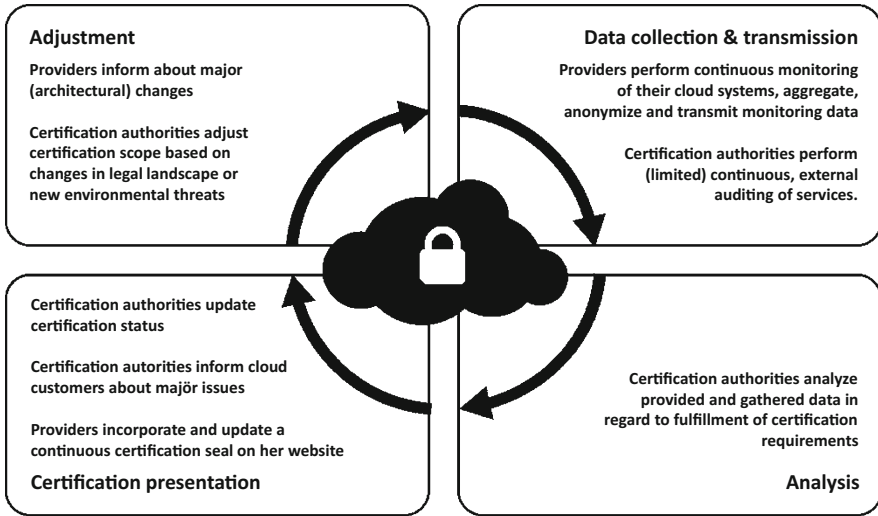


Fig. 7.4 Continuous certification process (adapted from Lins et al. (2016a))

In contrast to annual surveillance audits of conventional certification, automated data collection, and analysis of certification-relevant data, enable certification authorities to actively detect and investigate critical defects as they occur. Such procedures also enable immediate reaction to changes or events concerning a cloud service, and help certification authorities to adjust their certification reports based on an ongoing assessment of these defects, changes, and events (Lins et al. 2019b). Through timely detection and continuous assurance of certification adherence as required in highly dynamic cloud service environments, CSC can improve the trustworthiness of issued certifications. Consequently, in contrast to conventional certification, CSC considers the actual *status quo* of the cloud infrastructure when they assess certification adherence, and the CSC ultimately informs cloud customers on both infrastructure improvements (i.e., better service quality) or failures (i.e., data losses) through a transparent and up-to-date certification representation.

Two distinct but complementary types of CSC exist, namely test-based and monitoring-based certification (Anisetti et al. 2017; Kunz and Stephanow 2017; Lins et al. 2018; Stephanow et al. 2016). Test-based CSC methodologies are operated by certification authorities to collect certification-relevant information by directly accessing the cloud service infrastructure and testing cloud service components (Stephanow and Banse 2017). Typically, test-based certification techniques produce evidence by controlling some input to the cloud service and evaluating the output, such as calling a cloud service's RESTful API and comparing responses with expected results (Kunz and Stephanow 2017). Test-based CSC can be applied to verify the integrity of multiple cloud users' data (Wang et al. 2014), to assess data location (Doelitzscher et al. 2012), or to validate adherence to security criteria, among others (Stephanow and Khajehmoogahi 2017). In contrast, monitoring-

based certification techniques use data collected during monitoring as evidence collected from components involved in service delivery the cloud service is being provided (Kunz and Stephanow 2017). Implementing a CSC monitoring system enables cloud providers to (continuously) monitor their cloud infrastructures, and in the process collect and then transmit certification-relevant information to certification authorities (Lins et al. 2018; Lins et al. 2019b). For example, a prototypical monitoring-based CSC infrastructure (called '*CUMULUS*') was developed to, for instance, verify database user identification that could validate certification criteria (Krotsiani et al. 2015).

Performing monitoring- or test-based CSC is beneficial for cloud providers, certification authorities, and cloud service customers alike (Lins et al. 2016b; Teigeler et al. 2018). Cloud providers receive ongoing third party expert assessment of their systems. This enables providers detect potential flaws and (security) incidents earlier than before, and can save costs due to successive service improvements. Certification authorities actively detect and investigate critical certification deviations as they occur, thus increasing certification reliability. These authorities can increase their auditing efficiency, achieve savings in their budgets, reduce operation times, and reduce operation fees by relying on automated auditing processes to reduce auditing time and errors. Similarly, compared to their manual predecessors, CSC is more cost-effective due to certification authorities being able to test larger samples and examine data faster and more efficiently. Cloud service customers can benefit from CSC as well. Typically, cloud environments lack control because cloud customers cede governance to cloud providers (ENISA 2012). CSC can counteract this lack of control by increasing transparency regarding providers' operations, and providing assurance regarding requirements (e.g., ensuring encryption, data integrity, and location), and so ultimately enhancing trustworthiness of cloud services. Informing customers in cases of critical certification violations or major security breaches is becoming increasingly important nowadays because organizations, individuals, and even societies and economies are highly dependent on cloud services during their day-to-day activities (Benlian et al. 2018).

Summary

As has been mentioned, cloud computing is an evolution of information technology and a dominant business model for delivering IT resources. With cloud computing, individuals and organizations can gain on-demand network access to a shared pool of managed and scalable IT resources, such as servers, storage, and applications. While cloud computing emerged from related computing paradigms, such as application service provision and grid computing, and it shares similarities with IT outsourcing and fog computing, there are several peculiarities that distinguish it from related paradigms. In particular, how individuals and organizations access and use IT resources has changed as a result of cloud computing's inherent characteristics, such as on-demand self-service, ubiquitous access, multitenancy, rapid

elasticity, pay-per-use, as well as various service models (i.e., SaaS, PaaS, IaaS) and deployment models (i.e., public, private, hybrid, community). Today, we rely heavily on cloud services in our daily lives, e.g., for storing data, writing documents, managing businesses, and playing games online. Additionally, cloud computing provides the infrastructure that has powered key digital trends such as mobile computing, the Internet of Things, big data, and artificial intelligence, thereby accelerating industry dynamics, disrupting existing business models, and fueling the digital transformation.

By harnessing a diverse set of technology components, including the Internet, data centers, virtualization, and multi-tenant technologies, cloud providers offer innovative cloud services that enable organizations to achieve a vast number of benefits and opportunities. These benefits comprise cost savings that accrue from, among other things, a pay-per-use procedure, greater flexibility, access to leading edge IT resources, skills, and capabilities, as well as from lower entry barriers to markets or innovative IT services, and from focusing on core capabilities. More importantly, cloud computing exhibits transformative mechanisms that enable truly innovative services and business models, ultimately leading to fundamental and large-scale innovations that benefit not only individuals and organizations, but also markets and societies. In particular, the transformative mechanisms enable a system to become an independent service with a defined service interface, which then can be bundled and recombined with other services to create a platform within an ecosystem of diverse stakeholders.

In spite of the above-mentioned benefits, customers also face a high degree of uncertainty when they use cloud services. In particular, customers lose control of outsourced IT resources, and face uncertainty about the location where their data is stored and processed. Likewise, customers can be concerned about whether the cloud service is available whenever it is needed, and will be easy to use. Cloud computing also challenges contemporary security and privacy risk assessment approaches. Security and privacy breaches, data loss, as well as insecure interfaces and APIs are just a few examples of security and privacy challenges that cloud providers face. A common strategy to reduce customers' security and data protection uncertainty and to signal trustworthiness and adequate risk prevention, is the adoption of certification processes. To address the juxtaposition of static certifications in an ever-changing and dynamic cloud service environment, researchers have recently started to investigate how to achieve continuous service certification that will increase certification reliability and trustworthiness.

Defying initial concerns, cloud computing has already become a critical IT infrastructure for almost every aspect of our everyday lives, and it will continue to transform the world we live in on multiple levels and in various ways. For example, cloud computing is essential for the growth of artificial intelligence, because most types of hardware do not have the capabilities to run artificial intelligence applications (i.e., machine learning or deep neural networks) efficiently. Also, while cloud computing improves processing speed and accuracy of artificial intelligence applications, artificial intelligence can also be used to operate and manage cloud computing in a more efficient way, as in workload scheduling in clouds. The increased

use of cloud computing further results in a greater demand for cloud professionals in the future. Students should become familiar with the cloud computing paradigm to compete in future job markets.

Questions

1. What are the main characteristics of cloud services?
2. How does PaaS differ from IaaS?
3. What are the major reasons for organizations moving into the cloud?
4. Which risks that cloud service customers face, have been addressed by cloud providers?
5. What are transformative mechanisms of cloud computing and how do they interact?
6. How can continuous cloud service certification overcome the issues associated with current certification processes?

References

- Ahmed M, Litchfield AT (2018) Taxonomy for identification of security issues in cloud computing environments. *J Comput Inf Syst* 58(1):79–88
- Anisetti M, Ardagna C, Damiani E, Gaudenzi F (2017) A semi-automatic and trustworthy scheme for continuous cloud service certification. *IEEE Trans Serv Comput* 10(1):1–14
- Annette JR, Banu WA, Chandran PS (2015) Rendering-as-a-service: taxonomy and comparison. *Procedia Comput Sci* 50:276–281
- Apps Run The World (2017) Cloud applications revenue from leading vendors worldwide in 2015 and 2016 (in million U.S. dollars). <https://www.statista.com/statistics/475844/cloud-applications-revenues-worldwide-by-vendor/>. Accessed 11 Dec 2018
- Arasaratnam O (2011) Introduction to cloud computing. In: Halpert B (ed) *Auditing cloud computing, a security and privacy guide*. Wiley, Hoboken, NJ, pp 1–13
- Bajaj A (2000) A study of senior information systems managers decision models in adopting new computing architectures. *J AIS* 1(1es):4
- Baldwin LP, Irani Z, Love PED (2001) Outsourcing information systems: drawing lessons from a banking case study. *Eur J Inf Syst* 10(1):15–24
- Bayramusta M, Nasir VA (2016) A fad or future of IT?: a comprehensive literature review on the cloud computing research. *Int J Inf Manag* 36(4):635–644
- Benlian A, Hess T (2011) Opportunities and risks of software-as-a-service: findings from a survey of IT executives. *Decis Support Syst* 52(1):232–246
- Benlian A, Kettinger WJ, Sunyaev A, Winkler TJ (2018) Special section: The transformative value of cloud computing: a decoupling, platformization, and recombination theoretical framework. *J Manag Inf Syst* 35(3):719–739
- Bennett C, Timbrell GT (2000) Application service providers: will they succeed? *Inf Syst Front* 2(2):195–211
- Berman F, Hey T (2004) The scientific imperative. In: Foster I, Kesselman C (eds) *The grid 2: blueprint for a new computing infrastructure*. The Morgan Kaufmann series in computer architecture and design. Morgan Kaufman, San Francisco, CA, pp 13–24

- Bharadwaj A, El Sawy OA, Pavlou PA, Venkatraman N (2013) Digital business strategy: toward a next generation of insights. *MIS Q* 37(2):471–482
- Bhattacharjee A, Park SC (2014) Why end-users move to the cloud: a migration-theoretic analysis. *Eur J Inf Syst* 23(3):357–372
- Böhm M, Leimeister S, Riedl C, Krcmar H (2011) Cloud computing – outsourcing 2.0 or a new business model for IT provisioning? In: Keuper F, Oecking C, Degenhardt A (eds) *Application management: challenges – service creation – strategies*. Gabler, Wiesbaden, pp 31–56
- Bunker G, Thomson D (2006) *Delivering utility computing: business-driver IT optimization*. Wiley, Chichester
- Burda D, Teuteberg F (2014) The role of trust and risk perceptions in cloud archiving — results from an empirical study. *J High Technol Manag Res* 25(2):172–187
- Burnham TA, Frels JK, Mahajan V (2003) Consumer switching costs: a typology, antecedents, and consequences. *J Acad Mark Sci* 31(2):109
- Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Futur Gener Comput Syst* 25(6):599–616
- Cafaro M, Aloisio G (2011) Grids, clouds, and virtualization. In: Cafaro M, Aloisio G (eds) *Grids, clouds and virtualization. Computer communications and networks*. Springer, London, pp 1–21
- Carr N (2008) *The big switch: rewiring the world, from Edison to Google*. W.W. Norton, New York, NY
- Cloudscene (2018) Top ten data center operators in North America, EMEA, Oceania and Asia for the January to March 2018 period. <https://cloudscene.com/top10>. Accessed 11 Dec 2018
- Cusumano M (2010) Cloud computing and SaaS as new computing platforms. *Commun ACM* 53(4):27–29
- Dašić P, Dašić J, Crvenković B (2016) Service models for cloud computing: search as a service (SaaS). *Int J Eng Technol (IJET)* 8(5):2366–2373
- Dillon T, Wu C, Chang E (2010) Cloud computing: issues and challenges. Paper presented at the 24th IEEE international conference on advanced information networking and applications, Perth, WA, 20–23 Apr 2010
- Doelitzscher F, Fischer C, Moskal D, Reich C, Knahl M, Clarke N (2012) Validating cloud infrastructure changes by cloud audits. Paper presented at the 8th IEEE world congress on services, Honolulu, HI, 24–29 June 2012
- Durkee D (2010) Why cloud computing will never be free. *Commun ACM* 53(5):62–69
- ENISA (2012) Cloud computing – benefits, risks and recommendations for information security. European network and security agency. <https://resilience.enisa.europa.eu/cloud-security-and-resilience/publications/cloud-computing-benefits-risks-and-recommendations-for-information-security>. Accessed 17 Sept 2019
- Fernandes DAB, Soares LFB, Gomes JV, Freire MM, Inácio PRM (2014) Security issues in cloud environments: a survey. *Int J Inf Secur* 13(2):113–170
- Foster I, Kesselman C (2004) *The grid 2: blueprint for a new computing infrastructure*, 2nd edn. Elsevier, San Francisco, CA
- Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. *Int J High Perform Comput Appl* 15(3):200–222
- Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared. Paper presented at the grid computing environments workshop, Austin, TX, 12–16 Nov 2008
- Gao F, Thiebes S, Sunyaev A (2018) Rethinking the meaning of cloud computing for health care: a taxonomic perspective and future research directions. *J Med Internet Res* 20(7):e10041
- Gartner (2018a) Revenue of cloud computing worldwide. <https://de.statista.com/statistik/daten/studie/195760/umfrage/umsatz-mit-cloud-computing-weltweit/>. Accessed 9 Dec 2018
- Gartner (2018b) Revenues from public cloud infrastructure as a service (IaaS) market worldwide from 2015 to 2017, by vendor (in million U.S. dollars). <https://www.statista.com/statistics/754826/worldwide-public-cloud-infrastructure-services-vendor-revenues/>. Accessed 11 Dec 2018
- Gefen D, Karahanna E, Straub DW (2003) Trust and TAM in online shopping: an integrated model. *MIS Q* 27(1):51–90

- Gonzalez R, Gasco J, Llop J (2009) Information systems outsourcing reasons and risks: an empirical study. *Int J Soc Sci* 4(3):180–191
- Grover V, Kohli R (2012) Cocreating IT value: new capabilities and metrics for multifirm environments. *MIS Q* 36(1):225–232
- Grozev N, Buyya R (2014) Inter-cloud architectures and application brokering: taxonomy and survey. *Softw Pract Exp* 44(3):369–390
- Hentschel R, Leyh C, Petznick A (2018) Current cloud challenges in Germany: the perspective of cloud service providers. *J Cloud Comput* 7(1):5
- Hess T, Matt C, Benlian A, Wiesböck F (2016) Options for formulating a digital transformation strategy. *MIS Q Exec* 15(2):123–139
- Höfer CN, Karagiannis G (2011) Cloud computing services: taxonomy and comparison. *J Internet Serv Appl* 2(2):81–94
- Hogendorn C (2011) Excessive(?) entry of national telecom networks, 1990–2001. *Telecommun Policy* 35(11):920–932
- Hussain SA, Fatima M, Saeed A, Raza I, Shahzad RK (2017) Multilevel classification of security concerns in cloud computing. *Appl Comput Inform* 13(1):57–65
- ISO (2004) Conformity assessment – vocabulary and general principles. <https://www.iso.org/standard/29316.html>. Accessed 17 Sept 2019
- ITCandor (2018) Distribution of cloud platform as a service (PaaS) market revenues worldwide from 2015 to June 2018, by vendor. <https://www.statista.com/statistics/540521/worldwide-cloud-platform-revenue-share-by-vendor/>. Accessed 11 Dec 2018
- Iyer B, Henderson JC (2010) Preparing for the future: understanding the seven capabilities of cloud computing. *MIS Q Exec* 9(2):117–131
- Jones MA, Mothersbaugh DL, Beatty SE (2002) Why customers stay: measuring the underlying dimensions of services switching costs and managing their differential strategic outcomes. *J Bus Res* 55(6):441–450
- Kern T, Lacity MC, Willcocks L (2002) *Netsourcing: renting business applications and services over a network*. Prentice-Hall, New York, NY
- Khan N, Al-Yasiri A (2016) Identifying cloud security threats to strengthen cloud computing adoption framework. *Procedia Comput Sci* 94:485–490
- Killalea T (2008) Meet the virts. *ACM Queue* 6(1):14–18
- Kleinrock L (2005) A vision for the internet. *ST J Res* 2(1):4–5
- Kouatli I (2014) A comparative study of the evolution of vulnerabilities in IT systems and its relation to the new concept of cloud computing. *J Manag Hist* 20(4):409–433
- Kroeker KL (2011) Grid computing's future. *Commun ACM* 54(3):15–17
- Krotsiani M, Spanoudakis G, Kloukinas C (2015) Monitoring-based certification of cloud service security. Paper presented at the OTM confederated international conferences “On the move to meaningful internet systems”, Rhodes, 26–30 Oct 2015
- Kumar N, DuPree L (2011) Protection and privacy of information assets in the cloud. In: Halpert B (ed) *Auditing cloud computing, a security and privacy guide*. Wiley, Hoboken, NJ, pp 97–128
- Kunz I, Stephanow P (2017) A process model to support continuous certification of cloud services. Paper presented at the 31st IEEE international conference on advanced information networking and applications, Taipei, 27–29 Mar 2017
- Lansing J, Benlian A, Sunyaev A (2018) ‘Unblackboxing’ decision makers’ interpretations of IS certifications in the context of cloud service certifications. *J Assoc Inf Syst* 19(11):1064–1096
- Lee C, Wan G (2010) Including subjective norm and technology trust in the technology acceptance model: a case of e-ticketing in China. *SIGMIS Database* 41(4):40–51
- Leimeister S, Böhm M, Riedl C, Krcmar H (2010) The business perspective of cloud computing: actors, roles and value networks. Paper presented at the 18th European conference on information systems, Pretoria, 7–9 June 2010
- Lins S, Grochol P, Schneider S, Sunyaev A (2016a) Dynamic certification of cloud services: trust, but verify! *IEEE Secur Priv* 14(2):66–71

- Lins S, Teigeler H, Sunyaev A (2016b) Towards a bright future: enhancing diffusion of continuous cloud service auditing by third parties. Paper presented at the 24th European conference on information systems, Istanbul, 12–15 June 2016
- Lins S, Schneider S, Sunyaev A (2018) Trust is good, control is better: creating secure clouds by continuous auditing. *IEEE Trans Cloud Comput* 6(3):890–903
- Lins S, Schneider S, Sunyaev A (2019a) *Cloud-Service-Zertifizierung: Ein Rahmenwerk und Kriterienkatalog zur Zertifizierung von Cloud-Services*, 2nd edn. Springer, Berlin
- Lins S, Schneider S, Szefer J, Ibraheem S, Sunyaev A (2019b) Designing monitoring systems for continuous certification of cloud services: deriving meta-requirements and design guidelines. *Commun Assoc Inf Syst* 44:460–510
- Linthicum DS (2009) *Cloud computing and SOA convergence in your enterprise: a step-by-step guide: how to use SaaS, SOA, Mashups, and web 2.0 to break down the IT gates*. Addison-Wesley, Boston, MA
- Liu F, Tong J, Mao J, Bohn R, Messina J, Badger L, Leaf D (2011) NIST cloud computing reference architecture. https://bigdatawg.nist.gov/_uploadfiles/M0008_v1_7256814129.pdf. Accessed 11 Dec 2018
- Luftman J, Zadeh HS (2011) Key information technology and management issues 2010–11: an international study. *J Inf Technol* 26(3):193–204
- Malluhi Q, Khan KM (2013) Trust in cloud services: providing more controls to clients. *Computer* 46(7):94–96
- Marston S, Li Z, Bandyopadhyay S, Zhang J, Ghalsasi A (2011) Cloud computing — the business perspective. *Decis Support Syst* 51(1):176–189
- Mell P, Grance T (2011) The NIST definition of cloud computing. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>. Accessed 17 Sept 2019
- NIST Cloud Computing Security Working Group (2013) NIST cloud computing security reference architecture. <https://csrc.nist.gov/publications/detail/sp/500-299/draft>. Accessed 10 Dec 2018
- Pahl C (2015) Containerization and the PaaS cloud. *IEEE Cloud Comput* 2(3):24–31
- Park S-T, Park E-M, Seo J-H, Li G (2016) Factors affecting the continuous use of cloud service: focused on security risks. *Clust Comput* 19(1):485–495
- Sarkar P, Young L (2011) Sailing the cloud: a case study of perceptions and changing roles in an Australian University. Paper presented at the 19th European conference on information systems, Helsinki, 9–11 June 2011
- Schneider S, Sunyaev A (2016) Determinant factors of cloud-sourcing decisions: reflecting on the IT outsourcing literature in the era of cloud computing. *J Inf Technol* 31(1):1–31
- Schneider S, Lansing J, Gao F, Sunyaev A (2014) A taxonomic perspective on certification schemes: development of a taxonomy for cloud service certification criteria. Paper presented at the 47th Hawaii international conference on system sciences, Waikoloa, HI, 6–9 Jan 2014
- Schneider S, Wollersheim J, Krcmar H, Sunyaev A (2018) How do requirements evolve over time? a case study investigating the role of context and experiences in the evolution of enterprise software requirements. *J Inf Technol* 33(2):151–170
- Schwarz A, Jayatilaka B, Hirschheim R, Gales T (2009) A conjoint approach to understanding IT application services outsourcing. *J Assoc Inf Syst* 10(10):1
- Schwiegelshohn U, Badia RM, Bubak M, Danelutto M, Dustdar S, Gagliardi F, Geiger A, Hluchy L, Kranzlmüller D, Laure E, Priol T, Reinefeld A, Resch M, Reuter A, Rienhoff O, Rüter T, Sloot P, Talia D, Ullmann K, Yahyapour R (2010) Perspectives on grid computing. *Futur Gener Comput Syst* 26(8):1104–1115
- Seethamraju R (2013) Determinants of SaaS ERP systems adoption. Paper presented at the 17th Pacific Asia conference on information systems, Jeju Island, 18–22 June 2013
- Sehgal NK, Sohoni S, Xiong Y, Fritz D, Mulia W, Acken JM (2011) A cross section of the issues and research activities related to both information security and cloud computing. *IETE Tech Rev* 28(4):279–291
- Sharma D, Dhote C, Potey MM (2016) Identity and access management as security-as-a-service from clouds. *Procedia Comput Sci* 79:170–174

- Singh A, Chatterjee K (2017) Cloud security issues and challenges: a survey. *J Netw Comput Appl* 79(C):88–115
- Singh S, Jeong Y-S, Park JH (2016) A survey on cloud computing security. *J Netw Comput Appl* 75(C):200–222
- Soares J, Carapinha J, Melo M, Monteiro R, Sargento S (2011) Building virtual private clouds with network-aware cloud. Paper presented at the 5th international conference on advanced engineering computing and applications in sciences, Lisbon, 20–25 Nov 2011
- Stephanow P, Banse C (2017) Evaluating the performance of continuous test-based cloud service certification. Paper presented at the 17th IEEE/ACM international symposium on cluster, cloud and grid computing, Madrid, 14–17 May 2017
- Stephanow P, Khajehmoogahi K (2017) Towards continuous security certification of software-as-a-service applications using web application testing techniques. Paper presented at the 31st IEEE international conference on advanced information networking and applications, Taipei, 27–29 Mar 2017
- Stephanow P, Banse C, Schütte J (2016) Generating threat profiles for cloud service certification systems. Paper presented at the 17th IEEE international symposium on high assurance systems engineering, Orlando, FL, 7–9 Jan 2016
- Subashini S, Kavitha V (2011) A survey on security issues in service delivery models of cloud computing. *J Netw Comput Appl* 34(1):1–11
- Sunyaev A, Schneider S (2013) Cloud services certification. *Commun ACM* 56(2):33–36
- Susarla A, Barua A, Whinston AB (2003) Understanding the service component of application service provision: an empirical analysis of satisfaction with ASP services. *MIS Q* 27(1):91–123
- Tan W, Fan Y, Ghoneim A, Hossain MA, Dustdar S (2016) From the service-oriented architecture to the web API economy. *IEEE Internet Comput* 20(4):64–68
- Teigeler H, Lins S, Sunyaev A (2018) Drivers vs. inhibitors – what clinches continuous service certification adoption by cloud service providers? Paper presented at the 51th Hawaii international conference on system sciences, Hilton Waikoloa Village, HI, 3–6 Jan 2018
- Thiebes S, Kleiber G, Sunyaev A (2017) Cancer genomics research in the cloud: a taxonomy of genome data sets. Paper presented at the 4th international workshop on genome privacy and security, Orlando, FL, 15 Oct 2017
- Tiwana A, Konsynski B, Bush AA (2010) Research commentary—platform evolution: coevolution of platform architecture, governance, and environmental dynamics. *Inf Syst Res* 21(4):675–687
- Top Threats Working Group (2016) The treacherous 12. Cloud computing top threats in 2016. https://downloads.cloudsecurityalliance.org/assets/research/top-threats/Treacherous-12_Cloud-Computing_Top-Threats.pdf. Accessed 17 Sept 2019
- Trenz M, Huntgeburth JC, Veit DJ (2013) The role of uncertainty in cloud computing continuance: antecedents, mitigators, and consequences. Paper presented at the 21st European conference on information systems, Utrecht, 5–8 June 2013
- Trenz M, Huntgeburth J, Veit D (2018) Uncertainty in cloud service relationships: uncovering the differential effect of three social influence processes on potential and current users. *Inf Manag* 55(8):971–983
- Venkatesh V, Morris MG, Davis GB, Davis FD (2003) User acceptance of information technology: toward a unified view. *MIS Q* 27(3):425–478
- Venters W, Whitley EA (2012) A critical review of cloud computing: researching desires and realities. *J Inf Technol* 27(3):179–197
- Voorsluys W, Broberg J, Buyya R (2011) Introduction to cloud computing. In: Buyya R, Broberg J, Goscinski A (eds) *Cloud computing*. Wiley, Hoboken, NJ, pp 3–42
- Wang B, Li B, Li H (2014) Oruta: privacy-preserving public auditing for shared data in the cloud. *IEEE Trans Cloud Comput* 2(1):43–56
- Weinhardt C, Anandasivam A, Blau B, Borissov N, Meinel T, Michalk W, Stöcker J (2009) Cloud computing—a classification, business models, and research directions. *Bus Inf Syst Eng* 1(5):391–399
- Whitman M, Mattord H (2011) Cloud-based IT governance. In: Halpert B (ed) *Auditing cloud computing: a security and privacy guide*. Wiley, Hoboken, NJ, pp 33–55

- Windhorst I, Sunyaev A (2013) Dynamic certification of cloud services. Paper presented at the 8th international conference on availability, reliability and security, Regensburg, 2–6 Sept 2013
- Yoo Y, Henfridsson O, Lyytinen K (2010) Research commentary—the new organizing logic of digital innovation: an agenda for information systems research. *Inf Syst Res* 21(4):724–735

Further Reading

- Benlian A, Kettinger WJ, Sunyaev A, Winkler TJ (2018) Special section: The transformative value of cloud computing: a decoupling, platformization, and recombination theoretical framework. *J Manag Inf Syst* 35(3):719–739
- Fernandes DAB, Soares LFB, Gomes JV, Freire MM, Inácio PRM (2014) Security issues in cloud environments: a survey. *Int J Inf Secur* 13(2):113–170
- Lins S, Schneider S, Sunyaev A (2019) Cloud-Service-Zertifizierung: Ein Rahmenwerk und Kriterienkatalog zur Zertifizierung von Cloud-Services, 2nd edn. Springer, Berlin
- Marston S, Li Z, Bandyopadhyay S, Zhang J, Ghalsasi A (2011) Cloud computing — the business perspective. *Decis Support Syst* 51(1):176–189

Chapter 8

Fog and Edge Computing



Abstract

Thanks to innovations like the Internet of Things and autonomous driving, millions of new devices, sensors, and applications will be going online in the near future. They will generate huge amounts of data, which connected technologies will have to be able to handle. Measuring, monitoring, analyzing, processing, and reacting are just a few examples of tasks involving the vast quantities of data that these devices, sensors, and applications will generate. Existing models like cloud computing are reaching their limits and will struggle to cope with this deluge of data. This chapter introduces fog and edge computing as a model in which computing power moves toward the sources where the data are generated. Following a brief definition and overview of fog and edge computing, eight of their unique characteristics are described, including contextual location awareness and low latency. Differences between this model and the better-known cloud computing model, as well as other related models, are also explained, and the challenges and opportunities of fog and edge computing are discussed. In addition to the definition and characteristics of fog and edge computing, examples of practical implementation are presented.

Learning Objectives of this Chapter

The main learning objective of this chapter is to get an understanding of the concept of fog and edge computing and the opportunities they offer to consumers and organizations. Understanding the differences between cloud computing, edge computing, and fog computing are also important key learnings of this chapter. The chapter describes key characteristics of each model and how these models can work together. The students will learn which challenges are important to consider when an organization or a consumer decides to use the fog or edge computing models for their use cases. In addition, the students will know how users can achieve benefit from applications in a fog or edge computing environment, and they will receive a few examples of real-world applications of fog and edge computing, as well as insights into the well-known OpenFog reference architecture of fog computing.

Structure of this Chapter

The first section introduces the concepts of fog and edge computing by defining the terms ‘fog computing,’ ‘fog nodes,’ ‘edge computing,’ and ‘mist computing,’ and

showing the potential for this new computing model. Other models like cloud computing will also be contrasted with fog and edge computing. The second section shows the challenges (security and synchronization problems) and opportunities (low latency, geographical awareness, and robustness) of fog and edge computing solutions in various application contexts. The last section illustrates the practical applications of fog and edge computing and introduces the OpenFog reference architecture.

8.1 Fog and Edge Computing Fundamentals

One of the biggest challenges of digital innovations like the Internet of Things (IoT), embedded artificial intelligence, ubiquitous computing, and 5G wireless networks is the managing, storing, and processing of huge amounts of data (Madsen et al. 2013; Brogi and Forti 2017). Millions of new devices, sensors, and applications fueled by these innovations will be going online in the next decade. They will generate huge amounts of data, which connected technologies will have to be able to handle. Measuring, monitoring, analyzing, processing, and reacting are just a few examples of tasks involving the vast quantities of data that these devices, sensors, and applications will generate. The trend of consuming and producing large amounts of data poses a challenge to existing models like cloud computing because they will struggle to cope with this imminent flood of data (Bittencourt et al. 2015). Another challenge is the high demand for low latency when providing the data. Latency refers to the total amount of time it takes to send an entire message from the sender to the receiver. Nowadays, cloud computing environments are offering access to data at any time and from everywhere. In the case of conventional cloud applications like Dropbox, low latencies have been given less consideration. In future applications, where computing models need to support applications in the IoT environment, the flow of data has to be optimal. For example, in the case of connected vehicles, it is essential to provide data in real time; thus, innovative ways are required to ensure low latency (Bonomi et al. 2012). Another challenge is data inaccessibility. In many use cases, the devices are unable to use an Internet connection with high bandwidth, or there is no connection available because the telecommunication providers still lack complete coverage, especially in rural areas. IoT devices need the data on time and face significant problems when the connection quality is poor. Rising costs and concerns with regard to third parties' handling of sensitive data are also challenges for existing Internet technologies like cloud computing. To cope with these challenges, fog and edge computing present new distributed architectures that help to reduce latency and support the storage, management, and processing of huge amounts of data (Bittencourt et al. 2015; Shi et al. 2016; Yi et al. 2015). Simply stated, fog and edge computing refer to the continuum between the cloud and each device that measures, monitors, analyzes, processes, or reacts to data from the cloud

ecosystem. There are a few differences between fog and edge computing: The fog computing architecture allows the distribution of core functions closer to the point where the data are generated or consumed. For example, a vehicle's sensor data may be processed nearby and used for reactions like a speed change of several nearby vehicles based on road traffic accidents. In this scenario, a vehicle could send sensor data to a nearby traffic light that includes fog computing elements. The data can be processed directly by this fog node inside the traffic light, and the resulting data can be sent back to the vehicle. The core functions are computing, storage, communication, controlling, and decision making. In the case of edge computing, in addition to moving these core functions closer to the devices that use them, the devices can also be integrated to serve these core functions (e.g., a smart device with sensors could support computing functions). This means that edge computing architecture even allows the distribution of core functions at the edge of the network where the data originated or are consumed, in contrast with fog computing, where the fog nodes process data nearby but not in the end-devices themselves (Varghese et al. 2016). Edge computing comprises numerous technologies that include sensor networks and mobile data acquisition (Shi et al. 2016). By using fog or edge computing, the consumer receives advantages like lower latency, improved location awareness, higher business agility, better support for mobility, and lower transportation costs (Mahmood 2018). A market estimate for the complete fog computing market (including transportation, smart cities, agriculture, and industry) is \$18.2 billion for 2022 (OpenFog Consortium 2017b). The market size of edge computing is expected to be around \$28.84 billion by 2025 (Grand View Research Inc 2019).

8.1.1 Definition and Characteristics of Fog Computing

Literature, online blogs, company white papers, and glossaries use various definitions of fog computing, and there is no consistent understanding of the term. Other terms like edge computing, mist computing, fogging, cyber foraging, and cloudlets are mentioned as synonyms, competing models, or supplementary models (Mahmood 2018; Kai et al. 2016). In 2019, there is still no agreed-on definition for fog computing. In this book, the National Institute of Standards and Technology's (NIST) definition for fog computing provides the basis for defining the term in this chapter (Iorga et al. 2018).

Fog Computing

"Fog computing is a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes (physical or virtual), residing between smart end-devices and centralized (cloud) services." (Iorga et al. 2018)

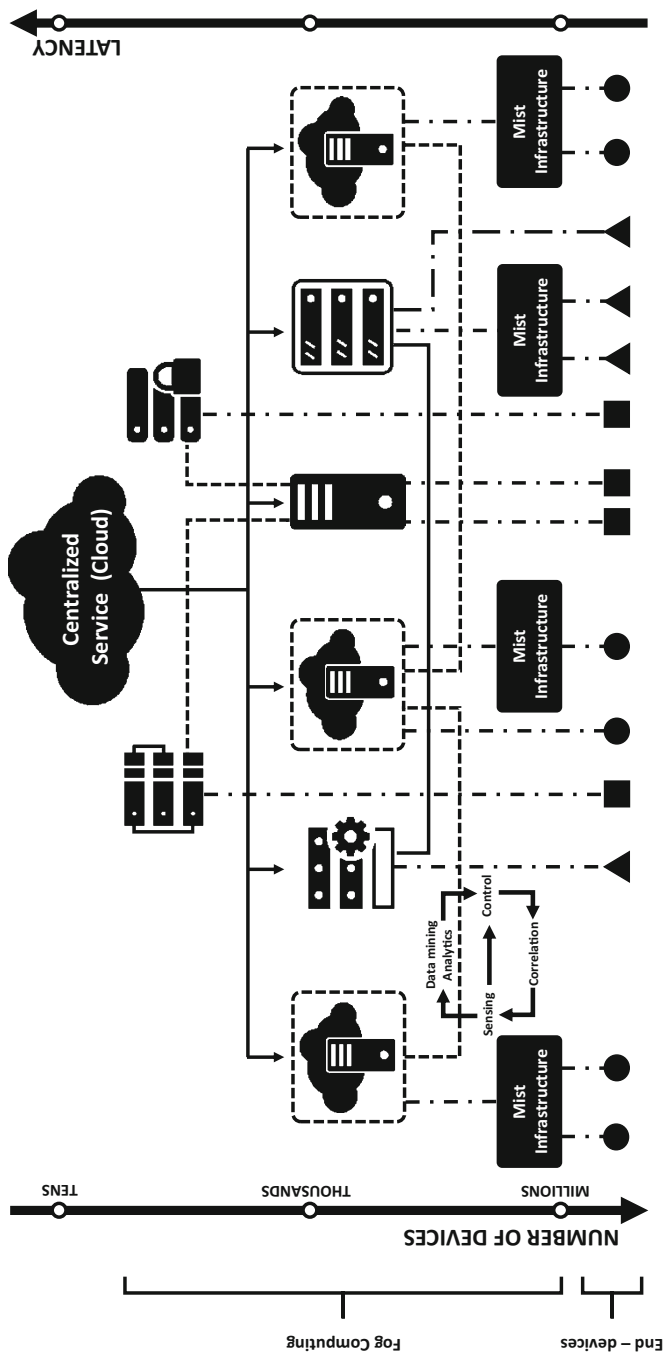


Fig. 8.1 Fog Computing Model (adapted from Iorga et al. (2017))

Fig. 8.1 shows the different layers that comprise a fog computing model. At the top is the traditional cloud computing layer. It offers services based on computing power in centralized data centers. At the bottom is all the end-devices. These devices can be sensors, actuators, or other smart devices and together they constitute the fog computing layer. It is a connection between large data centers and end-devices. However, this does not imply that these end-devices are dependent on the top layer. In many use cases of fog computing, the end-devices communicate with the so-called fog nodes in the fog computing layer and receive data that are computed, analyzed, or stored in the fog computing layer's fog nodes. Fig. 8.1 shows a use case where the fog computing layer supports the end-devices by offering several server functionalities (like computing) and the connection to the centralized services. Different use-case scenarios might have different architectures based on the requirements of the end-devices or the applications (Iorga et al. 2018). An edge computing layer, which is not included in the standard fog computing model, would be located between the sensors and the cloud, but it would be closer to the layer of sensors.

Fog nodes are aware of their geographical distribution and location within the context of a fog node cluster (sum of fog nodes that interact with each other). Additionally, fog nodes provide some form of data management and communication services between the network's edge, where the end-devices are, and the fog service or the centralized (cloud) computing resources, when needed. Usually, fog services are an extension of traditional cloud services as they make use of fog computing infrastructure to provide some of the services. To deploy a given fog computing capability, fog nodes operate in a centralized or decentralized manner. They can be configured as standalone fog nodes that communicate with each other to deliver the service or can be federated to form clusters that provide horizontal scalability over disparate geolocations through mirroring or extension mechanisms (Iorga et al. 2018).

Fog Nodes

“The fog node is the core component of the fog computing architecture. Fog nodes are either physical components (e.g., gateways, switches, routers, servers) or virtual components (e.g., virtualized switches, virtual machines, cloudlets, etc.) that are tightly coupled with the smart end-devices or access networks and provide computing resources to these devices.” (Iorga et al. 2018)

Fog nodes support one or more of the following key characteristics of computing devices: (1) autonomy, (2) heterogeneity, (3) hierarchical clustering, (4) manageability, and (5) programmability. Autonomy means that fog nodes can operate independently and make their own decisions locally. The different form factors (size and type of hardware) of fog nodes and the wide variety of environments are the reason for the heterogeneity. Fog nodes might also support hierarchical structures, different

layers, and several service functions but still interact as one continuum. Complex systems that can perform routine operations automatically manage the connected fog nodes (manageability). The programmability of fog nodes at multiple levels can be done by several stakeholders, for example, by network operators, domain experts, equipment providers, or even fog service consumers (Iorga et al. 2018).

The fog nodes are context-aware and support a common data management and communication system. They can be organized in clusters – vertically (to support isolation), horizontally (to support federation), or relative to fog nodes' latency distance to the smart end-devices. Fog computing minimizes the request-response time from/to supported applications and provides local computing resources for end-devices and, when needed, network connectivity to centralized services (Iorga et al. 2018).

Fog computing can be distinguished from other computing models by the following six key characteristics: (1) geographical distribution, (2) contextual location awareness and low latency, (3) heterogeneity, (4) interoperability and federation, (5) real-time interactions, and (6) scalability and agility of federated, fog clusters. In addition to these six key characteristics, the following two characteristics are often associated with fog computing: (7) predominance of wireless access and (8) support for mobility (Iorga et al. 2018; Yi et al. 2015; Bonomi et al. 2012). The following section describes each of these characteristics in detail.

Geographical Distribution

In contrast to the location of centralized cloud computing servers, fog computing consists of many fog nodes that are distributed in the environment. Based on this distribution, the fog nodes can track the location of the end-devices to support mobility (Hu et al. 2017). This geographical distribution is demanded by applications and services that need low latency for real-time decision making and are location-based or have special security requirements (Kai et al. 2016). For example, moving vehicles can use fog nodes that are located at highways, in parking areas, or at traffic lights to exchange and process data (Iorga et al. 2018; Bonomi et al. 2012). Special security requirements could be met by avoiding data transfers across the Internet (in the worst-case scenario, to data centers around the world). This helps to exclude potential attacks based on transmission over the Internet. Data packages might just be sent to a fog node nearby – a few centimeters from the point where the data were gathered. This means that the data do not come into contact with other systems such as routers on the Internet.

Contextual Location Awareness and Low Latency

Contextual location awareness relates to the large-scale geographical distribution of fog nodes. A subset of the fog nodes is located close to each other, and they can easily locate every device in the vicinity. This location awareness can be used to address several (non)functional requirements of the IoT, such as mobility (Negash et al. 2018). It is very advantageous for fog computing that the devices are aware of their location because many applications are location-based services (LBS). The location of fog nodes can be hard-coded or autonomously defined by connecting

with known devices nearby that know their own location (Tammemäe et al. 2018). For example, a printer in the office may be aware of its location. Other connected devices in this area can determine their actual location based on location information from this printer. This location awareness is an important aspect of fog computing. It helps to achieve the lowest possible latency because it allows fog nodes to choose the shortest communication path between fog nodes based on the location information (Iorga et al. 2018; Bonomi et al. 2012). Fog nodes are often co-located with other smart devices, which means the response time for these devices is much faster than in a traditional cloud computing architecture in which the data are hosted in a data center at a different location (Hu et al. 2017).

Heterogeneity

An essential characteristic of fog computing is its heterogeneity. Fog nodes can be virtual or physical, and the functions of these nodes are also very different and can change very quickly (Kai et al. 2016). The functions of these fog nodes can range from storing data to analyzing or processing big amounts of data. The architecture of fog computing also supports different form factors (from large to small devices) acquired through different kinds of network communication features (Iorga et al. 2018). Because of these different form factors, fog nodes are likely to have more limited capabilities than large servers in the context of cloud computing. There might also be big differences between fog nodes in other application areas and even between fog nodes in the same domain (Mouradian et al. 2018). Another aspect that leads to the heterogeneity of fog computing is the variety of distributed environments (office environment vs. outdoor environment) in which it is used (Mahmood and Ramachandran 2018). This heterogeneity needs to be taken into account when deciding which applications to deploy in which fog computing environment (Mouradian et al. 2018).

Interoperability and Federation

The seamless support of certain services (e.g. in the context of streaming services) requires cooperation between different fog service providers (Vatanparvar and Al Faruque 2018). Fog computing infrastructures have to be able to operate with each other, and services must be federated across different application areas (Bonomi et al. 2012; Iorga et al. 2018). In general, the interoperability aspect of fog computing means that each fog node can provide and use services from other actors in the fog infrastructure and utilize these services to operate effectively together (OpenFog Consortium 2017a). Based on the possibility that applications and services can be executed from different fog nodes (sometimes from other fog service providers), it implies the need for appropriate signaling and control interfaces, as well as appropriate data interfaces to enable interoperability at the level of providers and architectural modules. More precisely, control interfaces are needed to enable interactions between the different domains that are involved to support the application's lifecycle (Mouradian et al. 2018). The fog computing model will lead to new forms of cooperation and competition between different fog service providers (Kai et al. 2016). This interoperability also poses new challenges regarding access control

management of the different actors using these heterogeneous fog resources (Fakeeh 2016).

Real-time Interactions

In contrast to other computing architectures like cloud computing, real-time interactions are one of the key characteristics of fog computing. Fog computing supports, for example, real-time decision making (and other latency-sensitive or time-sensitive applications) for real-time services without any interruptions (Madakam and Bhagat 2018; Bonomi et al. 2012), which is a very important functionality in the application area of connected cars. In the context of connected cars, the real-time interaction of using fog computing between vehicles, access points, and traffic lights makes it much safer and more effective (Mahmood and Ramachandran 2018). Another example is a smart healthcare application for monitoring and detecting heart problems. In these two examples, the classical approach of batch processing in a data center would not be appropriate (Iorga et al. 2018).

Scalability and Agility of Federated Fog Clusters

At its core, fog computing is scalable because of its different clusters of fog nodes or even clusters of clusters. These clusters support elastic computing, resource pooling, data-load changes, and several network conditions, among others (Iorga et al. 2018). The high scalability of fog computing gives the consumer the illusion of infinite resources in the backend (similarly to cloud computing) (Bermbach et al. 2017). Many organizations starting to use fog computing will want to start modestly. They should avoid too much of an initial capital expense and grow that same initial network seamlessly and without disruption to serve complex web-scale services with millions of users (Byers 2017). Fog computing also enhances scalability by implementing modular fog nodes. In this context, modularity means that fog nodes build on a modular hardware platform. This allows the use of additional computing, network, and storage modules that can be added once the load on specific fog nodes increases (Byers 2017). Concerning the scalability of the network itself, the fog computing architecture allows load balancing between several fog nodes. When a fog node reaches its maximum computing load, the nearest fog nodes in the network will help to perform the computing tasks. These mechanisms let the fog computing model compute each task immediately without any additional latencies due to a single overburdened fog node.

From a software perspective, fog nodes can start rather small (the basic fog platform software and just a few applications) and grow gradually as more applications are added. Licensing models could also permit the incremental activation of fog node capacity through software as the network's demands grow (Byers 2017). Another important aspect of fog computing is the agility offered by supporting a variety of different use cases of applications. These include applications from different contexts like IoT, health care, or connected vehicles. In some scenarios, a mix of these applications is deployed on the same fog cluster or even fog nodes. This agility is an important advantage of fog computing because cloud computing solutions struggle to handle a mix of many different applications (Luan et al. 2015).

Predominance of Wireless Access

Fog computing is also used in a wired environment, but the large scale of wireless sensors in IoT implementations demand distributed computing power. Because there is a high demand for wireless solutions, fog computing is well-suited to meet this demand (Iorga et al. 2018). The type of connection ranges from high-speed links connecting enterprise data centers and the different wireless access technologies like 3G, WiFi, LTE, ZigBee, Bluetooth, or even 5G, which connects many mobile devices (Bonomi et al. 2012).

Support for Mobility

Many fog computing applications have to communicate directly with mobile devices. Therefore, fog computing generally needs to support standards for mobile devices. Fog computing can support the Locator/ID Separation Protocol, which decouples the identity of a device from its current location and requires a distributed directory system (Iorga et al. 2018). The mobility aspect is one of the key distinctions between fog computing and cloud computing (Mahmood and Ramachandran 2018).

8.1.2 Fog Computing Service Models

Similarly to cloud computing, fog computing defines three service types: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) (Iorga et al. 2018; Yi et al. 2015). Each fog node adheres to exactly one service type. For this reason, they are also referred to as fog node architectural service types (Iorga et al. 2018). The differences between these models for service provisioning are described below.

Software-as-a-Service

In the SaaS service type, a fog service provider offers fog applications directly to consumers (Mahmood 2018). These applications run on a cluster of fog nodes that are managed by the fog service provider. The consumer can access these services through a thin client interface or a program interface by using an end-device. There is no way for the consumer to control or adjust the application's underlying infrastructure and platform aspects. This means the consumer is unable to change settings regarding the network, server, storage, operating system, or general application. The user can configure limited, application-specific aspects. This limitation on configuration by the user leads to a highly standardized application and a very limited possibility to customize the consumed services.

Platform-as-a-Service

The PaaS service model offers the fog services consumer the opportunity to run and deploy their applications on a platform provided by the fog service provider (Mahmood 2018). These applications can be self-developed or acquired

applications. One limitation on flexibility concerns the use of libraries, programming languages, services, and tools supported by the underlying platform. The consumer has no means to control or adjust the platform's underlying infrastructure. This means the consumer is unable to change the network, server, storage, operating system, or general application settings. Unlike with the SaaS service model, the consumer can manage the deployed applications and the application-hosting environment. Instead of using defined standard applications, which SaaS consumers can do, the PaaS model gives more flexibility in the customization and administration of implemented applications.

Infrastructure-as-a-Service

The IaaS service model provides a high degree of flexibility to the fog service consumers because they are able to adjust all aspects built on top of the infrastructure of the fog service (Mahmood 2018). In contrast to the SaaS and PaaS service models, the consumer here is able to manage the operating system, storage, network, and server. The consumer might also be able to manage infrastructure elements like firewall settings. As in the PaaS service model, the consumers can deploy and run any applications, although they are not allowed to manage the underlying fog node cluster.

8.1.3 Fog Computing Deployment Models

Comparable to the fog computing service types, the deployment models also rely on existing cloud computing deployment models. The fog computing deployment models are divided into private fog, community fog, public fog, and hybrid fog (Iorga et al. 2017; Yi et al. 2015; Fakeeh 2016). Each fog computing deployment model can be offered by just one fog node so that they also can be called fog node deployment models (Iorga et al. 2017). In the following section, the characteristics of these different deployment models are briefly outlined.

Private Fog

Private fog is the most commonly used deployment model of fog computing. Fog service providers configure a fog infrastructure within a device that can only be used by a specified organization (Chang et al. 2017). These organizations can have many consumers using the fog node. This fog node can be owned, managed, or operated by the organization, a third party, or some combination of both (Iorga et al. 2018), and it can be located either on- or off-premise. Off-premise fog nodes operate on dedicated hardware that happens to be in a different building than the consumers. Organizations have the same (remote) access to the fog nodes as in an on-premise solution. Unlike off-premise services, an on-premise fog service is the operation of fog nodes in the organization's own building(s). Organizations can benefit in many ways from the private fog model – for example, they retain greater control over their data than in public or hybrid fog models because they do not share fog nodes with other consumers.

Community Fog

The community fog deployment model is client-capable infrastructure shared by consumers of organizations with shared concerns, interests, or requirements. These shared requirements may include compliance with regulations or industry requirements such as the ISO 27001 IT-security standard or can relate to performance requirements, such as real-time applications that require special techniques. Another shared requirement could relate to the mission or security. The goal of a community fog is to offer participating organizations the same benefits as a public fog (Iorga et al. 2018). Examples are multi-client capability and billing by usage. In addition, however, there is better data protection, security, and policy compliance, which are normally only guaranteed in private fogs. As in the private fog model, the fog nodes can be on- or off-premise.

Public Fog

In the public fog model, fog service providers make fog services available to any paying consumer via generic platforms (Chang et al. 2017). Public fog offers can be free of charge or charged according to actual usage. Subscription models are also possible in this context and may be owned by a business, academic, or government organization (Iorga et al. 2018). This organization also manages and operates the covered fog nodes. As in the private fog model, the fog nodes can be on- or off-premise.

Hybrid Fog

The hybrid fog model combines two or more distinct fog node deployment models (private, public, or community) (Iorga et al. 2018). This hybrid approach allows organizations to take advantage of the scalability and cost-effectiveness that a public fog offers without having to hand over business-critical data and applications to third parties. Since a hybrid fog includes both public and private (or even community) fog elements, the deployment of a hybrid fog can be planned from both. When introducing a hybrid fog, the primary goal should always be to change as little as possible of the existing IT infrastructure. No matter how similar the architecture of public, community, and private fog models may be, there will always be differences. However, the greater the differences in the fog architecture are, the harder it will be to manage the different fogs as a single environment.

8.1.4 Definition and Characteristics of Edge Computing

In the literature, the difference between fog computing and edge computing is not clear-cut. Edge computing focuses on the “things” side, while fog computing focuses more on the infrastructure side (Shi et al. 2016). Some literature describes edge computing as a synonym for fog computing (Hao et al. 2017; Mukherjee et al. 2017), but there are vital differences (Iorga et al. 2018).

Edge Computing

“Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services.” (Shi et al. 2016)

Edge computing describes the layer of end-devices that are used to do some local computing or sensor metering (Iorga et al. 2018). The combination of hardware and software of the end-devices in the edge computing environment is called edge nodes. It stands in stark contrast to the fog computing layer, which is based on fog nodes and not on end-devices themselves. Fog computing runs applications in a multi-layered architecture that provides the hardware and software functions, which allows dynamic reconfigurations for different applications while performing intelligent computing and transmission services. Edge computing runs specific applications in a fixed logic location and provides a direct transmission service. Fog computing is hierarchical, whereas edge computing tends to be limited to a small number of peripheral devices. Moreover, in addition to computation and networking, fog computing addresses storage, control, and data-processing acceleration (Iorga et al. 2018).

Edge computing is about processing data streams at least partially on the spot (e.g. directly at the end-device or within a factory) in a resource-saving way, while still benefiting from the advantages of fog and cloud computing. This approach requires the use of resources that are not permanently connected to a network, such as controllers, notebooks, smartphones, tablet computers, and sensors. Edge computing includes numerous technologies such as sensor networks, mobile data capture, mobile signature analysis, peer-to-peer, and ad hoc networking. Edge computing plays a major role in networking production equipment in Industry 4.0 with information and communication technology. Since the core properties of edge computing are almost identical to those of fog computing, they are not explored in this section. The main difference is the proximity to the sensors. Edge nodes are much closer to the sensors than fog nodes (Shi et al. 2016; Varghese et al. 2016). Thus, edge nodes are mainly used by just a single user. Fog nodes are mostly used by several users, devices, or applications at the same time – but not by a large number as in classic cloud computing.

A simple example of edge computing is a smartphone that is connected to various sensors on the user’s body. The sensors could be a heart rate monitor and a pedometer integrated into the shoes. In this case, the smartphone would be the edge node that aggregates, processes, and analyzes the data (Shi et al. 2016). In this example, there could be a direct connection between a smartphone and a cloud service. However, an additional fog layer could also be used and operates between the edge node and the cloud computing services. If, for example, the user goes jogging, the fog nodes integrated into traffic lights or the surrounding buildings are in the vicinity of the user. These fog nodes could take over other services that help to utilize the data collected in the edge node. In this scenario, a sensor request could be

processed directly by the edge node, the fog node, or even the cloud computing layer, depending on the type of request. This would dispense with a continuous data connection between the sensor and the cloud layer. Highly sensitive data such as the user's pulse would not be sent to the cloud but processed locally. This would minimize the latency and reduce or eliminate the load on the network.

8.1.5 Mist Computing

Mist computing is another term that is often used interchangeably with edge and fog computing in the literature. Mist computing is understood as a rudimentary, lightweight subset in fog computing. It builds a layer near the edge of the network and brings fog computing closer to smart devices. In mist computing, microcontrollers or microcomputers are used to forward data to fog nodes.

Mist Computing

“Mist computing is a lightweight and rudimentary form of fog computing that resides directly within the network fabric at the edge of the network fabric, bringing the fog computing layer closer to the smart end-devices. Mist computing uses microcomputers and microcontrollers to feed into fog computing nodes and potentially onward towards the centralized (cloud) computing services.” (Iorga et al. 2018)

The main goal of mist computing is to provide flexibility and manageability and reduce latencies even more than the standard approach of fog computing by pushing the processing tasks of fog nodes closer to the edges of the network (Tammemäe et al. 2018). Similarly, the nodes of mist computing involve sensors and actuator devices (Preden et al. 2015). One advantage of the proximity to the end-devices is resilience against communication instabilities because the distance between fog nodes and end-devices is kept to a minimum. These characteristics are often discussed with respect to outdoor applications like intelligence surveillance and connected vehicles (Tammemäe et al. 2018). In other words, mist computing enhances the fog computing approach to be nearly autonomous from other connections or systems by providing an architecture that allows devices to communicate directly with each other without any external communication channel like the Internet.

8.1.6 Differences to Cloud Computing

For organizations or consumers, the usage-based payment of the cloud computing model is an efficient alternative to owning and managing private data centers

(Bonomi et al. 2012; Hao et al. 2017). Several factors contribute to the economy of scale of these centralized data centers of cloud providers: higher predictability of massive aggregation, which allows higher utilization without degrading performance; a convenient location that takes advantage of inexpensive power; and lower operational expenditure achieved by deploying homogeneous computational, storage, and networking components. Cloud computing frees organizations and consumers from specifying many details.

This advantage becomes a problem for latency-sensitive applications that require nodes in the closed environment to meet their delay requirements. An emerging wave of new technologies, most notably the IoT, requires mobility support and geographical distribution in addition to location awareness and low latency (Bonomi et al. 2012).

To close this gap, fog computing can supplement the traditional cloud computing approach. Fog computing architectures also include traditional cloud computing elements like large data centers. So fog computing could be seen as an add-on that “that can bring the cloud applications closer to the physical end-devices at the network edge” (Tordera et al. 2016). While fog and cloud computing use the same resources (networking, compute, and storage) and share many of the same mechanisms and attributes (virtualization, multi-tenancy), the extension is a non-trivial one in that there are some fundamental differences stemming from the fog’s right to exist (Bonomi et al. 2012). Fig. 8.2 depicts the cooperation between cloud and fog computing, including the different communication types.

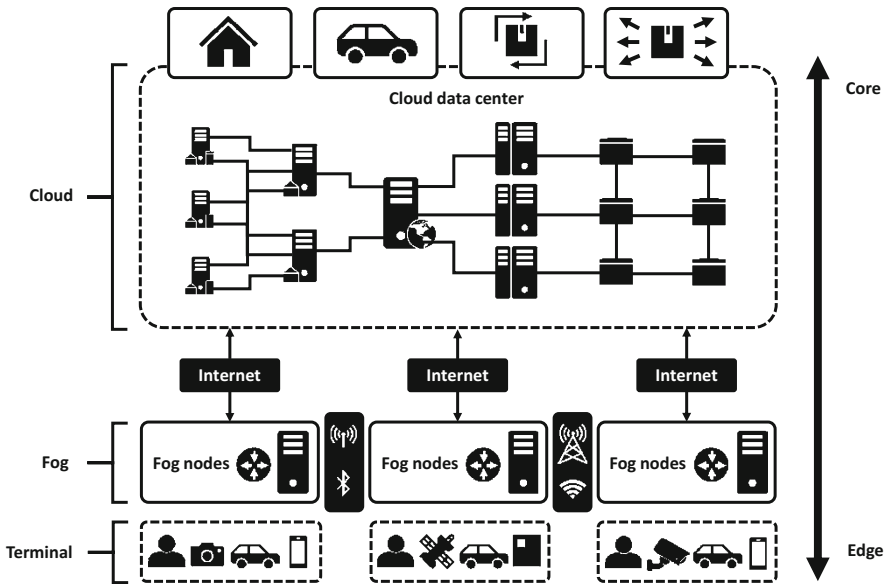


Fig. 8.2 Cloud and Fog Computing (adapted from Hu et al. (2017))

Table 8.1 summarizes the key differences between cloud and fog (or even edge) computing. In the section following this table, the cloud computing challenges that are met by fog and edge computing are discussed.

Table 8.1 Key differences: Cloud Computing vs. Fog and Edge Computing

	Cloud Computing	Fog and Edge Computing
Architecture	Centralized (servers in data centers)	Distributed (via fog or edge nodes)
Location of server nodes	On the Internet (global)	At the edge of the network (local)
Location awareness	No	Yes
Latency	Medium	Low
Amount of nodes	Few (large servers)	Millions (of fog or edge nodes)
Support for mobility	Limited	Supported
Computing capabilities	Higher	Lower
Analysis	Long-term/deep analysis	Short-term
Connection	Internet	Various protocols and standards
Risk of failure	Medium (depending on the Internet)	Low (based on different connections)
Security	Medium	High (due to distributed architecture)

For new application areas like the IoT or other new technologies, the traditional approach of cloud computing faces several challenges that can be met by using fog computing as a facilitator. While fog computing does not replace the structure of cloud computing at its core, fog computing might add a new layer that allows fog nodes to fulfill the requirements of these new applications. The challenges of cloud computing include latency requirements, geographical diversity, network bandwidth, reliability, robustness, and data protection concerning location-based services (LBS). Owing to its distributed architecture, fog computing offers fog nodes at the edges of a network. This means an end-device can communicate on a very short path with a device that can support computing (or other server functionalities) with the lowest latency level. Another challenge of cloud computing is the support of mobile applications with data-rich usage, meaning that comprehensive data sets have to be sent over the Internet. This might include several routers, network hops, and even wireless connections like 4G or 3G (Hu et al. 2017). These connections are not capable of handling vast amounts of data. With fog computing, the fog layer offers the opportunity to avoid such low capacity connections and deliver computing power and data from fog nodes in the nearby environment. These fog nodes can also act as local caches offering data that are hosted in the traditional cloud computing environment (Byers 2017).

Another aspect that challenges cloud computing is geographical diversity. Fog computing can handle the requirements of new applications that need their data, for example, to make real-time decisions. In the context of cloud computing, the whole network would be utilized in communication between data centers and end-devices. In the context of fog computing, the traffic load is limited to the short paths in the fog layer. The robustness and reliability of the offered services and data streams is another challenge for cloud computing and is handled by using fog computing.

The cloud computing approach fails when the end-device has no connection to the Internet or the data center is offline. In the context of fog computing, the fog nodes that are located close to the end-devices offer connections via different connection types and implement fast fail-over techniques (Madsen et al. 2013). The ability to make local responses without a connection to the Internet (in the case of cloud computing, it is the connection to the data centers) enables fog computing to increase the reliability and robustness of the services. User data privacy – especially its geographical data – is another challenge for cloud computing that could be solved with fog computing. In cloud computing, the servers have to know the location of each end-device to offer LBS. Fog computing could aggregate many end-devices in one fog node's area and anonymize this data or even keep it locally in the aggregated form.

8.2 Challenges and Opportunities of Fog and Edge Computing

While fog and edge computing have many advantages over the traditional model of cloud computing, they also have some weaknesses. In this section, the general challenges and opportunities of fog and edge computing are described. The aspects that are mentioned are on a general level and describe some examples of different application areas. Owing to the diversity with regard to domains of application, some of the mentioned aspects might not matter in a given context or may even be inverse. For example, some perceived security opportunities might be security challenges in other contexts or perspectives. Generally speaking, these aspects apply to edge and fog computing in the same way and may vary according to the composition of the specific use case. In particular, the aspects mentioned may have a different meaning depending on the definition of and demarcation between fog and edge computing.

8.2.1 Challenges of Fog and Edge Computing

Security Challenges

In general, fog and edge computing are distinguished from cloud computing by their distributed architecture. This distribution has many advantages, but from a security perspective, it is not optimal to deploy fog and edge nodes outside safe data centers. The fog and edge nodes are not in securely protected data centers with several security techniques like alarm systems, air conditioning, and redundant power supply. These data centers are usually certified to ensure a specific security standard in the context of physical risks. Fog and edge nodes are usually deployed in places without strict controls and protection capabilities (Hu et al. 2017).

The first major security challenge of fog and edge computing is trust. Fog and edge computing provide a distributed computing capability and build up such a

network in which every object has to communicate with the other extraneous objects (Aazam et al. 2018). Therefore, trust needs to be created in a fog and edge environment so that the collaboration between fog and edge nodes and end-devices can be executed successfully in practice (Martin et al. 2019; Garcia Lopez et al. 2015). Two extraordinarily important roles of trust in the fog or edge network are established in the literature: First, fog and edge nodes should be able to verify whether the requesting devices are reliable, and second, the end-devices that send the requests should also be able to identify whether the fog or edge nodes that are involved are secure at all (Mukherjee et al. 2017). However, the heterogeneous nature of fog and edge nodes makes the pursuit of trust even more difficult (Varghese et al. 2016).

There are three basic requirements of trust in fog and edge networks, namely the dynamic trust caused by the ongoing behavior variations of objects and topology of the fog or edge; the subjective trust depending on the characteristics of each object; and the context-dependent trust (Martin et al. 2019). In short, fostering trust between end-devices and fog or edge nodes plays a considerable role in preserving the security and reliability of fog and edge services (Alrawais et al. 2017).

Another critical security aspect of fog and edge computing is authentication and access control. Authentication plays a major role in maintaining the security of fog and edge computing, as front fog nodes supply various kinds of services to the end-users (Yi et al. 2015; Polito et al. 2009). It is necessary for each device to be identified as part of the network by authenticating itself to the fog or edge network so that it can gain access to the services (Mukherjee et al. 2017). Authentication in fog computing remains a considerable challenge, since the majority of fog nodes, including power, processing, and storage, are resource-constrained (Mukherjee et al. 2017). Therefore, traditional authentication mechanisms using public key infrastructure cannot be applied in these cases with fog or edge computing (Martin et al. 2019). Authentication at different levels of fog nodes is the primary security issue of fog computing (Stojmenovic and Wen 2014). Therefore, in the future, we have to pay more attention to ensure lightweight and end-to-end authentication (Mukherjee et al. 2017). Access control is a trustworthy technique to ensure the system is secure and user privacy is maintained by allowing authorized entities to access a specific resource (Alrawais et al. 2017). The massive number of objects and the geographically widely distributed data pose challenges to achieving access control.

Network security in fog and edge computing caused by wireless security issues is drawing significant public attention (Yi et al. 2015; Varghese et al. 2016). One of the most common issues is intrusion detection. Intrusion detection systems are those techniques that are applied in a cloud system to mitigate attacks such as insider attack, flooding attack, port scanning, and attacks on VM and hypervisor (Modi et al. 2013). In the context of fog and edge computing, intrusion detection systems are placed on the side of the fog or edge node system to detect intrusive behavior by monitoring and analyzing log files, access control policies, and user login information; at the same time, they can also be applied on the fog network side to anticipate malicious attacks (Modi et al. 2013; Yi et al. 2015). One representative instance is

the Denial-of-Service attack, which is brought about by sending lots of irrelevant service requests simultaneously. Since the fog or edge nodes are made busy for a long time by those irrelevant requests, services might go offline (Yi et al. 2015). The main challenge in coping with these issues is to determine the best detection system design – one that can be put to use in a geo-distributed, large-scale, high-mobility fog computing environment to meet the low-latency requirement (Yi et al. 2015). Rogue Node Detection focuses on detecting those malicious nodes that pretend to be legitimate but seek to exchange and collect data for malicious purposes. Once those rogue nodes are connected, the adversary can operate the incoming and outgoing requests, collect or tamper with user data stealthily, and make it easier to launch further attacks or threats (Alrawais et al. 2017).

Privacy issues also draw enormous attention as a significant security factor of fog and edge computing. It is clear that privacy preservation in fog and edge computing is much more challenging (Khan et al. 2017). Unlike cloud computing, fog and edge nodes are located close to the end-users. Therefore, it is easier for them to collect sensitive data concerning the identity, usage of utilities, or location of the end-users. Furthermore, since the fog and edge nodes are distributed in a decentralized manner, thorough control seems to be difficult. Frequent communication among fog or edge computing environments also relates to privacy leakage (Mukherjee et al. 2017; Shi et al. 2016).

The privacy issues of fog computing can be classified into three categories: data privacy, usage privacy, and location privacy (Yi et al. 2015). The data that sensor devices transfer to fog nodes are often sensitive, while restricted fog nodes can cause a loss of data and security impairments to the whole ecosystem (Guan et al. 2018). Therefore, further development of techniques like homomorphic encryption is meaningful, which enables privacy-preserving aggregation at the local gateways without decryption (Lu et al. 2012). Usage privacy involves data that relate to the user's behavior patterns and – because of the characteristics of fog computing – can easily be captured by attackers if they gain access to the fog devices (Yi et al. 2015). Data from a smart meter reveal lots of information about a household, such as whether the homes are inhabited or not. Criminals can get hold of such sensitive information for malicious purposes. How to design a smart way of partitioning the application to ensure the offloaded resource usage while protecting private information remains a challenge (Yi et al. 2015). Location privacy consists of information about the end-users' geographical position and even their path trajectory to the fog nodes, which means location information may be uncovered by investigating the users' usage habits (Yi et al. 2015).

Heterogeneity

Fog and edge computing models have multiple components, including various cloud nodes, fog nodes, and end-devices (mainly edge nodes).

As mentioned above, the computational and storage capabilities of fog, edge, and cloud computing differ significantly from each other. Thus, an essential aspect of the combination of fog, edge, and cloud computing models is the decision-making process for applications. Such a decision must take into account the availability of resources for fog nodes and the latency to process the workload in the fog, edge, or

cloud. Possible solutions in the research field are under discussion; one example is the Gaussian Process Regression for Fog-Cloud Allocation mechanism for determining where to schedule a workload to be processed by taking the resource availability and the latency overhead into consideration (da Silva and da Fonseca 2018). Another solution is the module mapping algorithm for the efficient utilization of resources in the network infrastructure by deploying application modules in fog-cloud infrastructure for IoT-based applications (Taneja and Davy 2017).

There might also be significant variations in terms of nodes in different fog and edge domains and even the nodes in the same fog or edge domain (Mouradian et al. 2018). Even in the fog layer, there are at least five kinds of node configurations: fog servers, networking devices, cloudlets, base stations, and vehicles (Mahmood 2018). These hardware platforms have varying levels of computation and storage capabilities, run different kinds of operating systems, and load a variety of software applications (Hu et al. 2017). Steps should be considered to overcome the heterogeneity. For instance, the limitations of specific nodes should be considered in the models and operations of the algorithms; for example, in some cases, a caching algorithm should take account of storage limitations (Mouradian et al. 2018). The need for federation should also be considered because the fog computing model has geographically distributed deployment on a broad scale, where fog domains are owned and operated by different fog service providers. Thus, provisioning applications requires the federation of these different providers, which implies seamless cooperation among these providers to ensure proper coordination of the necessary interactions between application components (Mouradian et al. 2018).

Furthermore, data are generated from different end-devices whose software, hardware, and technology interacts significantly with each other. All these components of the different layers are equipped with various kinds of processors and are not employed for general purpose computing (Mahmud et al. 2018). So it is critical for the fog or edge system to be able to manage the heterogeneity at such a large scale of difference. Ignoring the early standard development could cause several problems. Hence, this fast growth has led to an increase in heterogeneity and, in turn, to new problems such as vendor lock-in. This means that, because of dissimilarities among underlying architectures and programming languages, the code and data cannot (easily) be moved from one cloud to another, which causes further problems, particularly data integrity, interoperability, and portability (Sanaei et al. 2014). Recent research has focused on heterogeneity-related issues in mobile networks. For instance, a work-sharing approach called Honeybee is being developed to balance out independent jobs among heterogeneous mobile nodes (Fernando et al. 2019). Similarly, a framework based on service-oriented utility functions has been proposed to manage different resources that share tasks (Nishio et al. 2013).

Lastly, the network infrastructure of fog and edge computing is also heterogeneous, including not only high-speed links to the data center but also wireless access technologies such as WLAN, WiFi, 4G, ZigBee, etc. (Hu et al. 2017). So it is necessary to consider how the network should be aggregated to enable joint management with computing and storage resources. The current research direction is to manage the fog network using software-defined networks (SDN) or network

functions virtualization (Vaquero and Roderio-Merino 2014; Chin et al. 2015). In these scenarios, the fog node includes an SDN-like controller handling the programmability of the network of edge devices under the fog node control. Communication between different fog nodes can be handled through traditional routing mechanisms such as Open Shortest Path First and by following either fully distributed or centralized management using an SDN approach.

Programming Platform

In cloud computing, the infrastructure is transparent to the user, and the computation is done by the software (written in a specific programming language) that is deployed in the cloud. However, in fog and edge computing, the computation is done in user end-devices that most likely run on heterogeneous platforms and usually differ from each other. Thus, programming in such heterogeneous platforms is a huge challenge, and the need for a unified development framework for fog and edge computing has become indispensable.

Energy Management

Fog and edge computing systems consist of many distributed nodes. Thus, energy consumption is expected to be higher than for their cloud counterparts (Shi et al. 2016). For this reason, much work needs to be done to develop and optimize new and effective energy-saving protocols and architectures in fog or edge systems, such as efficient communications protocols, computing, and network resource optimization.

8.2.2 Opportunities

Low Latency

Low latency is one of the most critical characteristics of fog and edge computing. Fog and edge nodes at the edge of the network are responsible for locally processing and storing data collected by sensors and devices to carry out real-time analysis and have latency-sensitive applications. This operating mode can significantly reduce data movement across the Internet and provide speedy high-quality services at the local level (Hu et al. 2017). Therefore, fog computing is more suitable to provide services for latency-sensitive applications.

A reduction in latency does not happen automatically. The location of application components plays an important role as well (Mouradian et al. 2018). Algorithms can also help to reduce latency by managing resource utilization accordingly, such as in the area of task scheduling with minimum latency, devices offloading at the lowest latency, and content cache with latency considerations (Mouradian et al. 2018).

Scalability

As defined in the previous section, diverse requirements on fog and edge computing have resulted in the development of a specific fog and edge computing architecture. A large number of fog and edge nodes are generated to meet the demand of wide

geographical distribution (Wang et al. 2015). Nevertheless, fog computing is designed to cover millions of end-devices, which involves many applications and fog nodes (Mouradian et al. 2018). As a consequence, fog computing should be able to handle such large scales and prepare for elastic scalability at any time.

In contrast to cloud computing, fog computing tries to overcome the difficulty of providing high network bandwidth by delegating the huge requirements to the fog layer (Guan et al. 2018). Owing to the replacement of the requirements, the total pressure on cloud computing is reduced, since each fog node merely needs to take care of a small part of the demand. In short, scalability is defined as a meaningful opportunity for fog and edge computing thanks to this new assortment of services, which provides various applications for end-devices at the edge (Alrawais et al. 2017).

Geographical Distribution, Location, and Context Awareness

Since the technologies in ubiquitous and pervasive computing have been rapidly developing in recent years, the whole world seems to be tightly connected. Fog and edge computing are making tremendous efforts in this regard. One of the salient characteristics of fog and edge computing is the dense geographical distribution, which includes the distribution of fog and edge nodes along highways, roadways, and cellular phone towers (Shropshire 2014; Feng et al. 2017). As a result, fog and edge computing provide the end-users many more opportunities to connect with the Internet through the distributed fog and edge nodes than is the case with traditional cloud computing.

Furthermore, thanks to the fog nodes located nearby, location-based services (LBS) face fewer obstacles when cooperating with fog computing (Guan et al. 2018). Fog computing supports LBS fulfillment by supplying the end-users with fog nodes that are closer, which is more convenient for the end-users. Since the end-users only need to send their requests to the fog, the location information can be kept in secret, which cuts down the heavy workload pressure on the cloud and preserves the users' location privacy. The distributed endpoints and fog nodes ensure localization and sustain endpoints at the edge of the network in fog computing (Wang et al. 2015). Therefore, context awareness can be ensured.

Mobility Support

As mentioned in the section above, fog computing has the characteristic of geographical distribution because fog nodes may appear everywhere, including along highways, roadways, and cellular phone towers (Shropshire 2014). Therefore, fog nodes can communicate directly with mobile devices and process the collected data to enable mobile data analytics (Martin et al. 2019). This characteristic is extremely helpful in many scenarios such as the IoT, smart city, and smart vehicle, especially when mobile devices have to be able to react to events immediately when they occur.

Owing to the increasing number of mobile devices, the mobility support characteristic of fog computing is widely applied in practice. Mobility support makes fog computing more competitive than other computing models in applications related to mobile devices.

Dynamic and Autonomy

The dynamic of fog computing architecture is one of the most important features. It can be divided into two categories: One is the dynamic fog computing network and applications, and the other is the dynamic fog computing orchestration. First, the state of a fog computing network is constantly changing. The support of fog computing on mobile devices through a radio access network has enabled the mobility of the fog nodes. Together with the on/off switching of IoT applications and fog nodes' potentially unreliable network connection with the fog, the dynamic nature of fog computing is formed (Jiang et al. 2018).

Second, the dynamic nature of fog computing will be an important part of its future success. Nevertheless, given the dynamic nature of fog computing, the dynamic in the fog computing orchestration is more complicated. Basically, because of the dynamic nature of the network structure and applications, it is necessary for a fog computing network to tackle the dynamic autonomously in different tasks (Okafor et al. 2018).

Load Balancing

The number of end-devices producing and consuming data is expected to increase exponentially over the next few years (Chiang and Zhang 2016). Considering the significant amount of data that smart devices generate and send to the cloud servers every day, the cloud servers will undoubtedly be overloaded. By adopting fog and edge computing, however, redundant data can first be processed and filtered out at a decentralized level before being transported to the cloud. Hence, the cloud servers' workload will be significantly reduced.

On the one hand, a fog layer can filter the data being sent to the cloud in order to ensure cloud process performance; on the other hand, specific tasks can be transmitted from the mobile devices to the fog or edge nodes to promote performance. For instance, if a mobile device has to process a compute-intensive face recognition task, it can offload the complex computational process to a surrogate in the fog or edge system. The surrogate will perform the matching task more efficiently and deliver the result to the mobile device with low latency because of the short geographical distance (Mouradian et al. 2018). This process can overcome resource constraints on mobile devices and save storage and battery life, and the task itself can be accomplished more efficiently.

Energy Efficiency

Additionally, fog computing is more energy-efficient than cloud computing systems in some use cases, which also means lower cost and CO₂ emissions. The results of some studies between the energy efficacy of running cloud-based applications and using fog servers have shown that energy consumption is much lower in fog systems due to the reduced overhead of online interaction on fog nodes (Sarkar and Misra 2016; Jalali et al. 2016). Nevertheless, it is also proved that not all fog architectures are more energy-efficient than their cloud computing counterparts because there is no rational connecting network (Jalali et al. 2016). Thus, it is important to find more efficient algorithms to reduce energy consumption in the future, such as new energy models for shared or unshared network equipment, a task scheduling model (Oueis et al. 2015), or offloading strategies (Ye et al. 2016).

8.3 Fog and Edge Computing in Practice

The following section shows the practical solutions and potential use cases of fog and edge computing. Additionally, it provides a summary of the OpenFog reference architecture as defined by the OpenFog Consortium (2017a), which includes leading IT organizations, universities, and several other stakeholders.

8.3.1 *OpenFog Reference Architecture for Fog Computing*

The OpenFog reference architecture is a system-level architecture that extends elements of computer, networking, and storage across the cloud to the edge of the network. This architecture serves a specific subset of business problems that cannot be successfully implemented by using purely cloud-based architectures or relying solely on intelligent edge devices. OpenFog should be regarded as a complementary addition to and an extension of the traditional cloud-based model where implementations of the architecture can reside in multiple layers of a network's topology. The goal of the OpenFog architecture is to facilitate deployments that highlight interoperability, performance, security, scalability, programmability, reliability, availability, serviceability, and agility. Proprietary or single-vendor solutions can result in limited supplier diversity, which can have a negative impact on market adoption, system cost, quality, and innovation.

8.3.2 *Video Analytics*

The widespread use of mobile phones and network cameras enables new use cases for video analytics. Because of long data transmission latency and privacy concerns, cloud computing is no longer suitable for applications requiring video analytics. Consider the example of finding a person lost in the city. Nowadays, different kinds of cameras are widely deployed in the urban area and in every vehicle. When a person is missing, it is very possible that this person can be captured by a camera. However, because of privacy issues and traffic cost, the data from the camera will usually not be uploaded to the cloud, which makes it extremely difficult to leverage the wide-area camera data. Even if the data were accessible on the cloud, uploading and searching a huge quantity of data could take a long time, which is not feasible when tracking down a missing person. With the fog and edge computing model, the request to locate a person can be generated from the cloud and pushed to everything in a target area. Each device (e.g. every smartphone) can perform the request and search its local camera data and only report the result to the cloud. In this model, it is possible to leverage the data and computing power of everything and get the result much faster than with traditional cloud computing.

8.3.3 *Augmented Reality Glasses*

In the future, small devices in everyday life will be smart and require input from external data sources. This is the case with augmented reality glasses. There are strong restrictions on the amount of computing power the device has available. However, since augmented reality has a high demand for computing power, this has to be provided by devices located nearby. This is where fog or even edge computing models can help to solve the problem. The data required for the glasses will be provided directly by the surrounding environment from the fog or edge nodes. Today's augmented reality glasses do not yet have a suitable form factor in practice, as the computing power is still integrated inside the glasses. Owing to the implementation of the computing power in the glasses, they are also very expensive. By using modern edge or fog computing models, augmented reality glasses could become wearable and affordable for users in the near future. Motion sickness, which leads to dizziness for the users of augmenting reality glasses when the contents can only be displayed with a delay (Detecon International GmbH 2019), can be prevented with a high-performance edge or fog node connection.

Summary

In the past few years, fog and edge computing have become important computing models to enable new technologies like the Internet of Things. Based on these innovations and other upcoming trends like autonomous driving, millions of new devices, sensors, and applications will be going online in the coming years. They will generate huge amounts of data that have to be handled by connected technologies like fog and edge. In the near future, fog and edge nodes will have to measure, monitor, analyze, process, and react. Fog computing can be distinguished from other computing models by the following six key characteristics: (1) geographical distribution, (2) contextual location awareness and low latency, (3) heterogeneity, (4) interoperability and federation, (5) real-time interactions, and (6) scalability and agility of federated fog clusters. In addition to these six key characteristics, the following two characteristics are also often associated with fog computing: (7) predominance of wireless access and (8) support for mobility.

The challenges of fog and edge computing include security challenges, heterogeneity, programming platform, and energy management. This distribution has many advantages, but from a security perspective, it is not optimal to deploy fog and edge nodes outside of safe data centers. The fog and edge nodes are not in securely protected data centers with several security techniques like alarm systems, air conditioning, and a redundant power supply. The computational and storage capabilities differ significantly between fog, edge, and cloud computing, which leads to high heterogeneity. Beyond these challenges, however, the fog and edge computing models also provide the following opportunities: low latency, scalability,

mobility support, dynamic, autonomy, load balancing, energy efficiency, geographical distribution, location, and context awareness. Low latency is one of the most critical characteristics of fog and edge computing. Fog and edge nodes at the edge of the network are responsible for processing and storing data collected by sensors and devices at the local level to carry out real-time analysis and have latency-sensitive applications.

Edge and fog computing create new opportunities for the usage of end-devices and computing services. Application areas like autonomous driving, Internet of Things applications, and video surveillance systems are just some of the examples that benefit from the new computing models. Especially in the example of video surveillance based on fog or edge computing, several advantages are taken into account. On the one hand, the data could be analyzed in the cameras' vicinity, which excludes transmission along other networks and reduces latency; on the other hand, personal data like faces or private areas like recordings from homes are not processed on third-party devices in large data centers. These and the other opportunities of fog and edge computing may help these new computing models play an important role in future developments.

Questions

1. What are the key characteristics of fog and edge computing?
2. What is the role of a fog node in the fog computing environment?
3. Why is fog computing an add-on for the traditional cloud computing model?
4. What are the differences between edge computing and fog computing?
5. What are the challenges of fog and edge computing?
6. At a minimum, how many fog nodes does a fog computing infrastructure require?
7. What are the typical application areas of fog and edge computing?
8. What are the main problems of upcoming trends like the IoT that could not be handled by traditional cloud computing models?
9. In the context of security, what are the benefits of fog and edge computing over cloud computing?
10. Why could fog computing make real-time decision-making processes at the end-devices better than through the traditional cloud computing approach?

References

- Aazam M, Zeadally S, Harras KA (2018) Fog computing architecture, evaluation, and future research directions. *IEEE Commun Mag* 56(5):46–52
- Alrawais A, Alhothaily A, Hu C, Cheng X (2017) Fog computing for the internet of things: security and privacy issues. *IEEE Internet Comput* 21(2):34–42
- Bermbach D, Pallas F, Pérez DG, Plebani P, Anderson M, Kat R, Tai S (2017) A research perspective on fog computing. Paper presented at the international conference on service-oriented computing, Malaga, 13–16 Nov 2017

- Bittencourt LF, Lopes MM, Petri I, Rana OF (2015) Towards virtual machine migration in fog computing. Paper presented at the 10th international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC), Krakow, 4–6 Nov 2015
- Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. Paper presented at the first edition of the MCC workshop on mobile cloud computing, Helsinki, 17 Aug 2012
- Brogi A, Forti S (2017) QoS-aware deployment of IoT applications through the fog. *IEEE Internet Things J* 4(5):1185–1192
- Byers CC (2017) Architectural imperatives for fog computing: use cases, requirements, and architectural techniques for fog-enabled IoT networks. *IEEE Commun Mag* 55(8):14–20
- Chang C, Srirama SN, Buyya R (2017) Indie fog: an efficient fog-computing infrastructure for the internet of things. *Computer* 50(9):92–98
- Chiang M, Zhang T (2016) Fog and IoT: an overview of research opportunities. *IEEE Internet Things J* 3(6):854–864
- Chin WS, Kim H-s, Heo YJ, Jang JW (2015) A context-based future network infrastructure for IoT services. *Procedia Comput Sci* 56:266–270
- da Silva RA, da Fonseca NL (2018) Resource allocation mechanism for a fog-cloud infrastructure. Paper presented at the IEEE international conference on communications (ICC), Kansas City, MO, 20–24 May 2018
- Detecon International GmbH (2019) Edge computing ermöglicht innovative Anwendungsfälle. <https://www.detecon.com/de/wissen/edge-computing-ermoeslicht-innovative-anwendungsfaelle>. Accessed 12 Sept 2019
- Fakeeh KA (2016) Privacy and security problems in fog computing. *Commun Appl Electron* 4(6):19293–19304
- Feng J, Liu Z, Wu C, Ji Y (2017) AVE: autonomous vehicular edge computing framework with ACO-based scheduling. *IEEE Trans Veh Technol* 66(12):10660–10675
- Fernando N, Loke SW, Rahayu W (2019) Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. *IEEE Trans Cloud Comput* 7(2):329–343
- Garcia Lopez P, Montresor A, Epema D, Datta A, Higashino T, Iamnitchi A, Barcellos M, Felber P, Riviere E (2015) Edge-centric computing: vision and challenges. *ACM SIGCOMM Comput Commun Rev* 45(5):37–42
- Grand View Research Inc. (2019) Edge computing market size worth \$28.84 billion by 2025 | CAGR 54%. <https://www.grandviewresearch.com/press-release/global-edge-computing-market>. Accessed 12 June 2019
- Guan Y, Shao J, Wei G, Xie M (2018) Data security and privacy in fog computing. *IEEE Netw* 32(5):106–111
- Hao Z, Novak E, Yi S, Li Q (2017) Challenges and software architecture for fog computing. *IEEE Internet Comput* 21(2):44–53
- Hu P, Dhelim S, Ning H, Qiu T (2017) Survey on fog computing: architecture, key technologies, applications and open issues. *J Netw Comput Appl* 98:27–42
- Iorga M, Feldman L, Barton R, Martin MJ, Goren N, Mahmoudi C (2017) Fog computing conceptual model (NIST SP 800-191). <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf>. Accessed 17 Sept 2019
- Iorga M, Feldman L, Barton R, Martin MJ, Goren N, Mahmoudi C (2018) Fog computing conceptual model (NIST SP 500-325). <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf>. Accessed 17 Sept 2019
- Jalali F, Hinton K, Ayre R, Alpcan T, Tucker RS (2016) Fog computing may help to save energy in cloud computing. *IEEE J Sel Areas Commun* 34(5):1728–1739
- Jiang Y, Huang Z, Tsang DHK (2018) Challenges and solutions in fog computing orchestration. *IEEE Netw* 32(3):122–129
- Kai K, Cong W, Tao L (2016) Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues. *J China Univ Posts Telecommun* 23(2):56–96
- Khan S, Parkinson S, Qin Y (2017) Fog computing security: a review of current applications and security solutions. *J Cloud Comput* 6(1):19

- Lu R, Liang X, Li X, Lin X, Shen X (2012) EPPA: an efficient and privacy-preserving aggregation scheme for secure smart grid communications. *IEEE Trans Parallel Distrib Syst* 23(9):1621–1631
- Luan TH, Gao L, Li Z, Xiang Y, Wei G, Sun L (2015) Fog computing: focusing on mobile users at the edge. *arXiv preprint. arXiv:150201815*:1–11
- Madakam S, Bhagat P (2018) Fog computing in the IoT environment: principles, features, and models. In: Mahmood Z (ed) *Fog computing: concepts, frameworks and technologies*. Springer, Cham, pp 23–43
- Madsen H, Burtshy B, Albeanu G, Popentiu-Vladicescu F (2013) Reliability in the utility computing era: towards reliable fog computing. Paper presented at the 20th international conference on systems, signals and image processing (IWSSIP), Bucharest, 7–9 July 2013
- Mahmood Z (2018) *Fog computing: concepts, frameworks and technologies*, 1st edn. Springer, Cham
- Mahmood Z, Ramachandran M (2018) Fog computing: concepts, principles and related paradigms. In: Mahmood Z (ed) *Fog computing: concepts, frameworks and technologies*. Springer, Cham, pp 3–21
- Mahmud R, Kotagiri R, Buyya R (2018) Fog computing: a taxonomy, survey and future directions. In: Di Martino B, Li K-C, Yang LT, Esposito A (eds) *Internet of everything: algorithms, methodologies, technologies and perspectives*. Springer, Singapore, pp 103–130
- Martin JP, Kandasamy A, Chandrasekaran K, Joseph CT (2019) Elucidating the challenges for the praxis of fog computing: an aspect-based study. *Int J Commun Syst* 32(7):e3926
- Modi C, Patel D, Borisaniya B, Patel H, Patel A, Rajarajan M (2013) A survey of intrusion detection techniques in cloud. *J Netw Comput Appl* 36(1):42–57
- Mouradian C, Naboulsi D, Yangui S, Glitho RH, Morrow MJ, Polakos PA (2018) A comprehensive survey on fog computing: state-of-the-art and research challenges. *IEEE Commun Surv Tutor* 20(1):416–464
- Mukherjee M, Matam R, Shu L, Maglaras L, Ferrag MA, Choudhury N, Kumar V (2017) Security and privacy in fog computing: challenges. *IEEE Access* 5:19293–19304
- Negash B, Rahmani AM, Liljeberg P, Jantsch A (2018) Fog computing fundamentals in the internet-of-things. In: Rahmani AM, Liljeberg P, Preden J-S, Jantsch A (eds) *Fog computing in the internet of things: intelligence at the edge*. Springer, Cham, pp 3–13
- Nishio T, Shinkuma R, Takahashi T, Mandayam NB (2013) Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. Paper presented at the 1st international workshop on mobile cloud computing and networking, Bangalore, 29 July 2013
- Okafor K, Ononiwu G, Goundar S, Chijindu V, Chidiebele U (2018) Towards complex dynamic fog network orchestration using embedded neural switch. *Int J Comput Appl* 40:1–18
- OpenFog Consortium (2017a) OpenFog reference architecture for fog computing. https://iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. Accessed 17 Sept 2019
- OpenFog Consortium (2017b) Size and impact of fog computing market. <https://www.openfogconsortium.org/wp-content/uploads/451-Research-report-on-5-year-Market-Sizing-of-Fog-Oct-2017.pdf>. Accessed 20 Jan 2019
- Oueis J, Strinati EC, Barbarossa S (2015) The fog balancing: load distribution for small cell cloud computing. Paper presented at the 81st IEEE vehicular technology conference (VTC Spring), Glasgow, 11–14 May 2015
- Polito SG, Chamania M, Jukan A (2009) Extending the inter-domain PCE framework for authentication and authorization in GMPLS networks. Paper presented at the IEEE international conference on communications, Dresden, 14–18 June 2009
- Preden JS, Tammemäe K, Jantsch A, Leier M, Riid A, Calis E (2015) The benefits of self-awareness and attention in fog and mist computing. *Computer* 48(7):37–45
- Sanaei Z, Abolfazli S, Gani A, Buyya R (2014) Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Commun Surv Tutor* 16(1):369–392
- Sarkar S, Misra S (2016) Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *Iet Netw* 5(2):23–29
- Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. *IEEE Internet Things J* 3(5):637–646

- Shropshire J (2014) Extending the cloud with fog: security challenges & opportunities. Paper presented at the 20th Americas conference on information systems, Savannah, GA, 7–10 Aug 2014
- Stojmenovic I, Wen S (2014) The fog computing paradigm: scenarios and security issues. Paper presented at the federated conference on computer science and information systems, Warsaw, 7–10 Sept 2014
- Tammemäe K, Jantsch A, Kuusik A, Preden J-S, Õunapuu E (2018) Self-aware fog computing in private and secure spheres. In: Rahmani AM, Liljeberg P, Preden J-S, Jantsch A (eds) *Fog computing in the internet of things: intelligence at the edge*. Springer, Cham, pp 71–99
- Taneja M, Davy A (2017) Resource aware placement of IoT application modules in fog-cloud computing paradigm. Paper presented at the IFIP/IEEE symposium on integrated network and service management (IM), Lisbon, 8–12 May 2017
- Tordera EM, Masip-Bruin X, García-Almiñana J, Jukan A, Ren G-J, Zhu J, Farre J (2016) What is a fog node a tutorial on current concepts towards a common definition. arXiv preprint. arXiv:161109193:1–22
- Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. *SIGCOMM Comput Commun Rev* 44(5):27–32
- Varghese B, Wang N, Barbhuiya S, Kilpatrick P, Nikolopoulos DS (2016) Challenges and opportunities in edge computing. Paper presented at the IEEE international conference on smart cloud (SmartCloud), New York, NY, 18–20 Nov 2016
- Vatanparvar K, Al Faruque MA (2018) Control-as-a-service in cyber-physical energy systems over fog computing. In: Rahmani AM, Liljeberg P, Preden J-S, Jantsch A (eds) *Fog computing in the internet of things: intelligence at the edge*. Springer, Cham, pp 123–144
- Wang Y, Uehara T, Sasaki R (2015) Fog computing: issues and challenges in security and forensics. Paper presented at the 39th IEEE annual computer software and applications conference, Taichung, 1–5 July 2015
- Ye D, Wu M, Tang S, Yu R (2016) Scalable fog computing with service offloading in bus networks. Paper presented at the 3rd IEEE international conference on cyber security and cloud computing (CSCloud), Beijing, 25–27 June 2016
- Yi S, Qin Z, Li Q (2015) Security and privacy issues of fog computing: a survey. Paper presented at the international conference on wireless algorithms, systems, and applications, Qufu, 10–12 Aug 2015

Further Reading

- Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. Paper presented at the first edition of the MCC workshop on mobile cloud computing, Helsinki, 17 Aug 2012
- Iorga M, Feldman L, Barton R, Martin MJ, Goren N, Mahmoudi C (2018) Fog computing conceptual model (NIST SP 500-325). <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf>. Accessed 17 Sept 2019
- Kai K, Cong W, Tao L (2016) Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues. *J China Univ Posts Telecommun* 23(2):56–96
- Mahmood Z (2018) *Fog computing: concepts, frameworks and technologies*, 1st edn. Springer, Cham
- Yi S, Qin Z, Li Q (2015) Security and privacy issues of fog computing: a survey. Paper presented at the international conference on wireless algorithms, systems, and applications, Qufu, 10–12 Aug 2015

Chapter 9

Distributed Ledger Technology



Abstract

Distributed Ledger Technology (DLT) is one of the most promising innovations in the field of information technologies with the potential to change organization and collaboration in the economy, society, and industry. This chapter introduces the technical background and use cases of distributed ledger technology. It presents the major innovations originating from distributed ledger technology since the introduction of the blockchain concept. Furthermore, cryptocurrencies' historical background as a driver of fully decentralized distributed ledgers is outlined from their origins in the 1990s until the blockchain concept's introduction in 2009. DLT's technical principles are introduced to provide a sound understanding. Subsequently, the functioning of distributed ledger technology is illustrated by means of the Bitcoin blockchain example, which was the first fully decentralized cryptocurrency to not require a trusted authority (i.e., banks). Thereafter, smart contracts and the idea of decentralized applications are explained. Selected use cases for distributed ledger technology's application are subsequently discussed. This chapter concludes with a discussion of the prevailing challenges in the field of distributed ledger technology.

Learning Objectives of this Chapter

In this chapter, students are given a basic understanding of Distributed Ledger Technology (DLT), its historical background, and application possibilities. The chapter therefore explains DLT's technical innovations since the introduction of the blockchain concept. For a better understanding of DLT's functionality in general, the different types of DLT designs are explained, as are the terms hash value, Merkle tree, digital signature, and consensus mechanism. Students should also be able to assign these terms' role to a blockchain concept and internalize the idea and use of smart contracts. Finally, they should understand the usefulness of DLT in real-world application areas, be able to mention selected use cases, and explain DLT's role in these use cases.

Structure of this Chapter

Section one introduces DLT's potential compared to other distributed databases. Subsequently, DLT is given a nomenclature to establish a common understanding of

its terms. The historic background of DLT's evolution concludes section one. Section two explains the technical foundations of distributed ledger technology, including hashing and public key infrastructure. Based on the DLT nomenclature and the technical foundations, the Bitcoin blockchain's function is explained. Subsequently, smart contracts are introduced as a useful extension of distributed ledgers. Finally, various application areas integrating distributed ledger technology are presented and the current challenges in distributed ledger technology discussed.

9.1 Background of Distributed Ledger Technology

Distributed Ledger Technology (DLT) is one of the most 'hyped' technologies of the last few years. The wide adoption of the cryptocurrency Bitcoin made blockchain – and DLT in general – a highly discussed topic in practice and research. A cryptocurrency is a digital asset designed to be used as a medium of exchange. By applying cryptographic techniques, transactions, and represented assets are safeguarded from manipulation and theft (Chohan 2017). This section sheds light on why DLT has the potential to change the business landscape and everyday life in dramatic ways (Kursh and Gold 2016). Furthermore, the section provides an overview of the historic background of the best-known DLT application, namely cryptocurrencies. Finally, a terminology is presented for DLT concepts and characteristics and important DLT characteristics are introduced.

9.1.1 Distributed Ledger Technology as a Game Changer

Storing and managing data are an integral part of many applications, which is why databases are applied. Data are mostly organized in relational databases with clearly defined tables and dependencies between those tables holding the relevant data for the application. Data organization in databases allows for four operations, also referred to as CRUD (Create, Read, Update, Delete) operations. First, new data are *created* in the database. Once created, data can be *read*, *updated*, or be *deleted*. All database operations perform a certain unit of work on the database. The relevant unit of work is called a transaction, whose execution ends with a commit that makes the changes to the database visible to its other users. Each transaction is executed in isolation, which enables database operations to be clearly differentiated. Database operations can therefore be reversed in case of a failure (e.g., a create operation being interrupted).

Three types of databases can be used to organize data: *centralized databases*, *decentralized databases*, and *distributed databases*. *Centralized databases* reside on a single storage device and can, therefore, be better maintained than those distributed across multiple, physically separated, storage devices. However, centralized databases have drawbacks in terms of, for instance, their availability and performance. A

system's availability (here, a database) describes the probability of a system being reachable at a random time and of it functioning correctly. Centralized databases can become a bottleneck if too many requests need to be processed during a specific period, which decreases the system's overall performance. In *decentralized databases*, there is no central storage; data are simply stored on multiple storage devices connected with one another, but usually located in different physical locations. A decentralized database's storage devices are organized in a hierarchical structure with a set of nodes communicating with a particular node, which can be a node of a superordinate set of nodes. Decentralized databases therefore incorporate multiple, hierarchically organized, centralized databases. *Distributed databases* are a well-known concept for increasing availability and avoiding performance issues. In distributed databases, replications of the data are stored across multiple, physically independent, storage devices. When a distributed database's storage device fails or cannot be accessed for a time, other, still operating storage devices can therefore respond to open requests and provide a similar result. Such a distribution can increase performance, because it allows requests to be distributed across the storage devices and be processed in parallel. A distributed database's storage devices do not (usually) follow a hierarchical structure, but instead form a mesh of connected storage devices, with each storage device connected to another. The following refers to a distributed database's storage device as a node.

Distributed Database

A distributed database is a type of database where data are replicated across multiple storage devices (nodes) with equal rights.

Although distributed databases are physically separated on different nodes, a distributed database's CRUD operations should always return the same results, no matter on which node the operation is performed. For instance, proceeding with a read operation on one database replication run by a particular node should return the same result when executing the same read operation on any other node of the distributed database. Consequently, owing to all the nodes' targeted consistency, distributed databases are considered logically centralized, but architecturally distributed (Buterin 2017). To achieve consistency between all the nodes, the stored data need to be identical on each of the distributed database replications, which makes distributed databases generally more complex than centralized database. If all the data are identical across all the nodes, the distributed database is in a *consistent state*, with the all nodes having eventually agreed on this state and, therefore, having reached consensus. To reach a consistent state, the stored data must be synchronized between a distributed database's nodes. Because the nodes are physically distributed, they must communicate with one another by means of a network, which could, however, lead to Byzantine failures occurring, thus complicating consensus. The term Byzantine failure is derived from the Byzantine Generals' Problem (Lamport et al. 1982), referring to actors having to agree on a mutual strategy to avoid

catastrophic failure, but some of the actors are unreliable. In computer science, there are multiple forms of Byzantine failures. First, a node (e.g., a computer or robot) can be ascertained as having crashed or not being reachable via the network, because the node no longer responds. Second, a monitoring system may be unable to determine a certain node's status, which happens when, for instance, this node crashes or when network failures result in inconclusive responses from the node. Third, nodes may pursue malicious intentions, such as trying to store incorrect data in the distributed database. To overcome the first and second Byzantine failures, protocols and algorithms (e.g., Paxos, RAFT) were introduced and applied to ensure nodes' reliable synchronization and consistency in distributed databases (Lamport and Massa 2004). These algorithms and protocols managing the synchronization between nodes are called consensus mechanisms, since they seek to achieve a consistent state across a distributed database's nodes.

Consensus Mechanism

A consensus mechanism is designed to achieve agreement on the respective state of replications of stored data between a distributed database's nodes under consideration of network failures.

Consensus mechanisms can negotiate a consistent state in a probabilistic or total finality-preserving way. Probabilistic consensus mechanisms only assume to a certain probability that the majority of the benign nodes has agreed on the relevant stored ledger's specific state. Over time, the likelihood increases that all of the nodes will eventually agree on this state, because subsequent operations build on the previous states. However, it may still be possible to change the state post hoc (Eyal and Sirer 2014). In contrast, consensus mechanisms, which ensure total finality, establish the ledger's definitive state on which all the nodes agree. The particular state on which the nodes have agreed cannot be changed. However, most consensus mechanisms in distributed databases fail to handle the third Byzantine failure. Consequently, a distributed database's consistent state can only be established in a controlled network with verified, honest nodes.

The distributed ledger is special type of distributed database comprising a ledger's local replications on several nodes. Ledgers only allow new data to be appended; consequently, deleting or updating appended data post hoc should be impossible. Furthermore, distributed ledgers assume the presence of malicious nodes, which requires the applied consensus mechanism to cope with the third Byzantine failure (malicious behavior of nodes) (Hileman and Rauchs 2017). To assure consistency in distributed ledgers, the third Byzantine failure becomes even more crucial where nodes may arbitrarily join or leave the network (Pass and Shi 2017). Once the Bitcoin blockchain was introduced, it provided a solution for the third Byzantine failure in distributed ledgers (Nakamoto 2008), resulting in no central authority being required to administrate the distributed ledger's contents and to distinguish between honest and malicious nodes.

Distributed Ledger

A distributed ledger is a type of distributed database that assumes the presence of nodes with malicious intentions. A distributed ledger comprises a ledger's multiple replications in which data can only be appended or read.

Owing to the application of game theory to consensus finding in distributed databases, DLT allows distributed ledgers to be run on arbitrary nodes whose providers are not necessarily known or trusted. Game theory is the “[...] study of mathematical models of conflict and cooperation of intelligent rational decision-makers” (Myerson 2004). Anyone can contribute to a distributed ledger and participate in the consensus mechanism to assure that malicious nodes cannot corrupt the stored data (see chapter 9.2.4). The reliable synchronization of a distributed ledger's dynamically changing set of nodes in the presence of all types of Byzantine failures is one of DLT's main innovations.

Distributed Ledger Technology

Distributed Ledger Technology (DLT) enables the realization and operation of distributed ledgers, which allow benign nodes, through a shared consensus mechanism, to agree on an (almost) immutable record of transactions despite Byzantine failures and eventually achieving consistency.

The use of cryptocurrencies (e.g., Bitcoin) was one of the main drivers of blockchain and DLT's invention. Customers' need to trust their bank and the banking system that has to return their money after they have deposited it, was predominantly the motivation for cryptocurrencies. For instance, during the financial crisis, Greek bank customers lost confidence in the banking system and tried to secure their money by withdrawing cash. The sudden mass cash withdrawals threatened the banks' liquidity, which led the Greek government to introduce several regulatory interventions to tackle the issue. In 2013, the financial crisis became dramatic for the Bank of Cyprus customers. Deposits at the Bank of Cyprus exceeding EUR 100,000 lost up to 60% of their value when the rest of the money was converted into bank shares (Hadjicostis 2013). Bitcoins, and the later cryptocurrencies (e.g., Ethereum¹), allow people to own digital assets, such as digital coins, without the need for institutions such as banks. No other party may spend the asset without the owner's consent, which renders a trusted third party, such as a bank, obsolete for certain services.

Digital assets can also be transferred from the asset owner to another user via transactions committed to the ledger. DLT transactions can be compared to bank transfers, which include (at least) a sender, a receiver, and a certain value being transferred. The sender and the receiver are represented by a unique character string,

¹<https://www.ethereum.org/>

which is generated by using cryptographic techniques such as hash functions (see 9.2.1) and public key cryptography (see 9.2.3). All users can receive and send transactions through their accounts. In the DLT context, a transaction history keeps a chronological list of all the transactions associated with an account. For example, the transaction history represents a bank account's history of payment transactions (see 9.3). Using cryptographic techniques, such as hash functions (see 9.2.1), the transaction history's integrity is protected from post hoc changes to each node of the distributed ledger. Furthermore, public key cryptography ensures that no other user can transfer values that belong to another account by means of digital signatures (see 9.2.3).

The double-spending problem is a well-known one in cryptocurrencies, referring to the same user spending the same digital asset (e.g., a coin) multiple times. In cryptocurrencies double spending would occur, for example, if Alice paid Bob one coin and then simultaneously used the same coin to pay Carl. With physical coins, double spending of the same coin is obviously impossible. In addition, preventing the double spending of digital coins is very challenging, because these coins are only represented by numerical values and are not bound to physical objects. To overcome the double-spending problem, nodes are commonly used to validate transactions and only such validated transactions are included in the distributed ledger.

During the last decade, it was realized that DLT did not only apply to cryptocurrencies. Distributed ledgers' structure and functionality were developed further to allow them to store additional data beyond assets' values. Consequently, even programs, also called smart contracts (see 9.4), can be stored inside transactions. Such smart contracts can be used to formalize business processes (or parts thereof), which makes DLT a promising technology to automate and speed up business processes while decreasing transaction costs. The former is possible due to DLT's potential to remove parties from the equation, thus requiring less coordination effort and possibly speeding up business processes. Since intermediaries can be excluded from several services, the fees for these services can be reduced. In supply chain management, for example, DLT allows individual products' provenance to be retraced by means of the data stored on a distributed ledger during manufacturing or transportation (Corkery and Popper 2018). In the financial sector, DLT projects, such as Ripple², aim to accelerate inter-bank transactions while decreasing the costs. Internet users' identity management can be improved by using DLT to provide a platform that will dynamically grant and revoke access to their personal data, thus making the use of personal data more transparent to users without a trusted intermediary and preventing the manipulation of access rights. In terms of Estonia's public administration, it was transformed into a *digital government* by moving all government-related transactions online and backing these by means of a distributed ledger³ that allows documents to be digitally signed and stored immutably. Paper-based documents can now be stored as indelible, digital entries. Government-related transactions include taxes, voting, and issuing Estonian ID

²<https://ripple.com/>

³<https://e-estonia.com/>

cards. The Estonian ID card's concept integrates DLT to grant the owner access to electronic services such as banking and medical prescriptions. Section 9.5 presents a more detailed selection of important DLT application areas.

9.1.2 History of Distributed Ledger Technology

DLT became popular with the public with the rise of the cryptocurrency Bitcoin, which was presented in a white paper published in 2008 under the pseudonym Satoshi Nakamoto (Nakamoto 2008). The large scale-adoption of Bitcoin during the last decade may make one assume that it was the first cryptocurrency. However, the idea of cryptocurrencies already appeared in the 1980s, introduced by Dr. David Chaum (Chaum 1983). Originally incentivized by the vision of establishing an anonymous and secure digital voting system, Chaum developed a cryptographic technique called blind signatures that allowed pseudonymity in data exchanges. Blind signatures made use of public cryptography, which meant each user has a publicly known public key and a secret private key. Blind signatures allowed one to sign data digitally with a private key. The digitally signed data could be used to prove that the data had originated from a certain identity, although only this identity's public key was known. It is therefore not necessary to know the sender's real identity to validate a message's originator (see 9.2.3). Based on his findings in the field of cryptography, Chaum envisioned an anonymous electronic payment system (Chaum 1983), subsequently founding the company DigiCash in 1989, which developed the cryptocurrency eCash that focused on payments' anonymity and untraceability by using blind signatures (Chaum 1983). eCash was designed to enable micropayments through cryptographically secured files that stored a certain monetary value and were generated in exchange for fiat money (e.g., U.S. dollars). Consequently, customers needed to first transfer funds from their relevant bank account to their eCash account. Thereafter the bank generated encrypted files storing a certain value that was also stored on the eCash customer's computer. A special utility program was provided to manage the encrypted files and execute payments. To avoid the double-spending problem, Chaum's concept assigned banks the responsibility of verifying that the eCash had not already been spent in other transactions. For example, when Alice sends an eCash value to Bob, Bob then sends the received eCash to the issuing bank. The issuing bank then verifies that the eCash has not already been used. Chaum's concept was, however, still dependent on intermediaries, such as the participating trusted authorities.

eCash was tested in the United States as a micropayment system from 1995 to 1998. The system was free for purchasers, but merchants had to pay a transaction fee. Nevertheless, a broad user base remained unconvinced of eCash's benefits, and credit cards were introduced instead. In Europe, financial institutions remained more interested in eCash, because credit cards were not widely used, and cash transactions were preferred. Several European financial institutions, such as Credit Suisse and Deutsche Bank, therefore adopted eCash in 1998. Despite eCash's success, DigiCash filed a Chapter 11 bankruptcy in 1998 and was finally sold for assets in

2002, with credit cards replacing eCash. Later, PayPal enabled the immediate exchange of assets from person to person and customer to merchant, and Chaum's ideas of payment anonymity almost fell into oblivion.

In 1996, Douglas Jackson and Barry Downey introduced the cryptocurrency e-gold, which Gold & Silver Reserve Inc. ran under the name e-gold Ltd. As the name suggests, gold backed the e-gold cryptocurrency. In 2004, e-gold was the first cryptocurrency to reach a critical mass of customers and merchants when it registered more than 1,3 million accounts (e-gold 2004). To use e-gold, users had to register an account on the e-gold web site, which allowed them to make instant e-gold transfers to other e-gold accounts (Hughes et al. 2007). E-gold also offered an application programming interface (API), which enabled e-gold payments' integration into various services and e-commerce platforms (Gold and Silver Reserve 1999). However, e-gold became a target of early phishing scams and malware. Inspired by the first known phishing attack against a financial institution in June 2001 (Cryptography 2005), another attack was used to defraud e-gold in 2003. In 2007, asset transfers via e-gold were suspended due to legal issues, such as money laundry. Nonetheless, e-gold was the first successful online payment system and paved the way for many technologies currently still used in e-commerce, for instance, secure communication (for payments) using SSL encryption and the flexible integration of payment services into external systems by providing an API.

In 1997, Adam Back created one of the first implementations of a Proof of Work (PoW) system, called hashcash. Cynthia Dwork and Moni Naor (Cynthia and Moni 1993) invented the PoW concept, which Markus Jakobsson and Ari Juels (Jakobsson and Juels 1999) later formalized. PoW is an economic measure (e.g., computational power) used to prevent service abuses, such as spamming or Denial of Service (DoS) attacks (Back 1997). DoS attacks are aimed at making so many requests for a service that it cannot handle them and eventually crashes. In PoW systems, a service requester must perform a certain task, like solving a certain mathematical problem, before being allowed to request a service. On the one hand, this task needs be moderately hard to prevent the service requester from making too many requests; on the other hand, the service provider needs to assess the task's output easily. PoW systems do not usually require the users to solve these tasks, as the terminal devices perform the required task automatically. hashcash showed that PoW could be used to reduce spamming and the threat of DoS attacks in practice. Currently, PoW plays a fundamental role in consensus mechanisms applied to distributed ledgers, such as Bitcoin and Ethereum.

In 1998, the computer scientist Wei Dai shared two protocol drafts for a digital currency called b-money. He was driven by the idea of crypto-anarchy, which Timothy C. May introduced (Dai 1998). A crypto-anarchy involves a government's permanent absence and postulates anonymity and censorship resistance in respect of, for example, payments. In this context, a system is considered censorship resistant, if there is no possibility of modifying or even blocking a third party's data. Payments should be made by using cryptographically secured cryptocurrency tokens. Such tokens represent a medium of exchange to allow people to cooperate with each other efficiently, and are often called coins in cryptocurrencies (Dai 1998). In the same year, Nick Szabo had the initial idea for the development of BitGold (Szabo 1997).

BitGold was never implemented but is considered the direct precursor of the Bitcoin architecture. Szabo's BitGold scheme applied PoW, because the users would use computer power to solve the cryptographic equations that the system assigned. In addition, Szabo aimed at eliminating intermediaries and tried to prevent double spending by implementing algorithmic and structural improvements. The latter were still a problem at that time, because previous solutions still depended on a trusted authority. In essence, the BitGold concept envisioned a fully decentralized cryptocurrency by replacing intermediary processes with automated processes. However, both b-money and BitGold failed to achieve large scale adoption.

In 2009, the Bitcoin blockchain was implemented and made accessible to the public. Bitcoin paved the way for a distributed ledger for digital money transfers, which can be considered the DLT generation⁴ 1.0. The Bitcoin blockchain was the first cryptocurrency on a distributed ledger to solve all three types of Byzantine failures and eliminate the need for intermediaries (Nakamoto 2008). Since Bitcoin builds on a distributed ledger, its underlying infrastructure is architecturally distributed and logically centralized. Furthermore, Bitcoin is decentrally governed, because individual parties can maintain each node in the Bitcoin blockchain (Buterin 2017). After several years, Bitcoin became highly popular and reached a maximum market value of more than USD 19,783.06 per coin on December 11, 2017 (Morris 2018). Besides executing transactions, the Bitcoin blockchain allows the use of a scripting language to develop scripts that, for instance, automatically undertakes pay-outs if multiple parties agree on this payout (Atzei et al. 2018). Although Bitcoin's scripting language provides the environment to apply a simple version of smart contracts, it is very limited, because it is not Turing complete⁵. Incentivized by the Bitcoin breakthrough, several foundations, such as the Ethereum Foundation, were established to develop blockchain further. Although the Bitcoin blockchain introduced the concept for the first fully decentralized cryptocurrency, the Bitcoin blockchain came with several constraints, such as low throughput (seven transactions per second on average), pseudonymity instead of real anonymity, and the above-mentioned limitations of the scripting language. The subsequent DLT generation 2.0 specifically addressed the latter issue in order to improve DLT's flexibility and applicability in applications by means of more powerful smart contracts. In 2015, the Ethereum Foundation introduced the Ethereum blockchain, which paved the way for the use of Turing complete smart contracts on a distributed ledger. The Turing complete environment of Ethereum – the Ethereum Virtual Machine (EVM) – allows the execution of smart contracts that can be developed in a high-level programming language (HLL) (Buterin 2018). An HLL is an abstraction from executable machine code and provides a natural language for program code development. HLLs

⁴The term blockchain generation is normally used. However, since blockchain can be considered a DLT concept, the more general term DLT generation is used.

⁵Turing completeness (named after the British computer scientist Alan Turing) describes a system's ability to simulate any other Turing machine. Turing completeness enables the use of loops in programming, which is the multiple execution of programing code until a defined condition is fulfilled. The concept of loops is important in programming to increase flexibility and to decrease code duplications.

generally facilitate program code development through the automation of certain areas of computing systems (e.g. memory management). Ethereum advanced the basic Bitcoin transaction structure with ERC-20, a more flexible standard for smart contracts. Ethereum is therefore not limited to using Bitcoin as a cryptocurrency, but also enables the development of applications that share a distributed ledger as a common backend to, for example, store data. The game CryptoKitties⁶ is a popular example of a DLT application, because it uses Ethereum as a backend. In the DLT generation 3.0, the DLT designs were advanced to meet business use case requirements, such as a high throughput, improved confidentiality, and increased flexibility in the development of DLT applications. The initial idea of DLT designs as a publicly available, fully distributed ledger that allows user pseudonymity has therefore changed. To keep a distributed ledger’s data private, several DLT designs, such as HyperLedger Fabric⁷, are run by a small set of identifiable organizations or companies. These DLT applications are predominantly tied to business applications, which demand more flexible tokens in order to store arbitrary data. By offering this, DLT designs are applicable to various domains, such as supply chain management (Li et al. 2017), health IT (Dagher et al. 2018), access management (Alansari et al. 2017), and identity management (Abbasi and Khan 2017).

9.1.3 Terminology in Distributed Ledger Technology

Based on the introduced definition of DLT (see chapter 9.1.1), it is clear that DLT is a very broad term that is often used interchangeably with blockchain. DLT includes different concepts, which differ in the way the distributed ledger organizes transactions (Yeow et al. 2018; Kannengießner et al. 2019). The remainder of this chapter briefly introduces different DLT concepts and how they differ in terms of their DLT designs, DLT properties, and DLT characteristics (see Fig. 9.1).

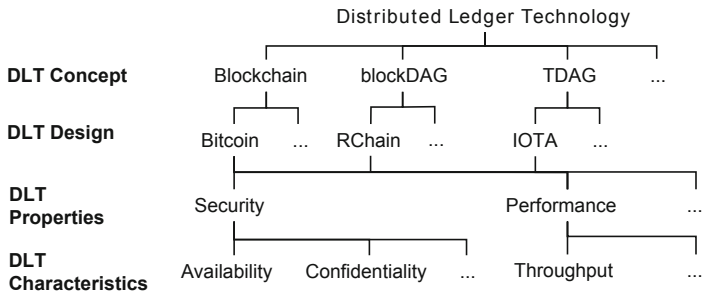


Fig. 9.1 Overview of the hierarchical structure of selected DLT concepts, DLT designs, DLT properties, and DLT characteristics

⁶<https://www.cryptokitties.co/>

⁷<https://www.hyperledger.org/projects/fabric>

A DLT concept provides an abstract description of a distributed ledger’s architecture and the organization of transactions. The DLT concepts that are most discussed are blockchain (Nakamoto 2008; Buterin 2018), block-based directed acyclic graphs (blockDAG) (Yeow et al. 2018; Sompolinsky et al. 2018), and transaction-based directed acyclic graphs (TDAG) (Yeow et al. 2018; Popov 2018). In blockchain, transactions are included in a superordinate data structure called blocks. The blocks are chained together, forming a chain of blocks, with each block cryptographically linked to a single predecessor. In contrast, blockDAGs can have multiple predecessors or successors. In TDAGs, blocks are not used at all and transactions are directly linked to others.

Each DLT concept can be implemented in different ways. The concrete implementation of a DLT concept is called a DLT design. For example, distributed ledgers’ Bitcoin and Ethereum both employ the DLT concept blockchain. However, Bitcoin and Ethereum’s implementations differ significantly. While Bitcoin generates a new block every 10 minutes on average, Ethereum issues a new block approximately every 7 seconds. Furthermore, Bitcoin comes with a pre-defined maximum block size of 1 MB. In contrast, Ethereum does not limit the size of blocks explicitly, but ties the costs in the form of coins to a particular block size. All DLT designs incorporate six DLT properties described in Table 9.1 (Kannengießer et al. 2019).

Table 9.1 DLT Properties (adapted from Kannengießer et al. (2019))

Development Flexibility
The possibilities that a DLT design offers for maintenance and further development.
Institutionalization
The emerging embedding of concepts and artifacts (here DLT) in social structures.
Anonymity
The degree to which individuals are not identifiable within a set of subjects (Pfitzmann et al. 2006).
Performance
The accomplishment of a given task measured against standards of accuracy, completeness, costs, and speed.
Security
Preservation of confidentiality, integrity, and the availability of information (ISO/IEC 2009).
Usability
The extent to which specified users can use a DLT design to achieve specified goals with respect to effectiveness, efficiency, and satisfaction in a use context (IOS/DIS 2018).

Each of the DLT properties incorporates multiple DLT characteristics, which are crucial for a DLT design’s suitability for DLT applications. Owing to the differences between DLT concepts, not every DLT characteristic applies to each of the DLT designs. For example, TDAGs will not feature in the DLT characteristic block size (Kannengießer et al. 2019). Table 9.2 presents an excerpt of often-mentioned DLT characteristics as summarized by Kannengießer et al. (Kannengießer et al. 2019). This chapter refers to the DLT characteristics as shown in Table 9.2.

Table 9.2 Excerpt of DLT Characteristics (adapted from Kannengießer et al. (2019))

Prop.	DLT characteristic
Performance	<i>Block creation interval</i>
	The time between consecutive blocks’ creation.
	<i>Scalability</i>
	A DLT design’s capability to handle an increasing amount of workload or its potential to be enlarged to accommodate that growth
	<i>Throughput</i>
	The number of transactions validated and appended to the ledger in a given time interval
Security	<i>Availability</i>
	The probability that a system operates correctly at an arbitrary point in time.
	<i>Integrity</i>
	DLT’s high degree of integrity requires stored transactions to be protected against unauthorized modification or deletion, as well as against authorized users’ irrevocable, accidental, and undesired changes.

9.2 Technical Foundations

DLT designs can be instantiated as a *public* or *private* distributed ledger (Xu et al. 2017; Yeow et al. 2018). In public DLT designs, the underlying network allows arbitrary nodes to join and participate in the distributed ledger’s maintenance. No registration or verification of the nodes’ identities is required. Public DLT designs are usually maintained by a large number of nodes, for example, in Bitcoin and Ethereum. Owing to the large number of nodes in the network, each of which stores a replication of the ledger, public DLT designs achieve a high level of availability. To allow many (arbitrary) nodes to find consensus, public DLT designs should be well scalable to not deter performance when the number of nodes increases. In contrast, private DLT designs engage a defined set of nodes, with each node identifiable and known to the other network nodes. Consequently, private DLT designs require verification of the nodes that join the distributed ledger, for example, by using a Public Key Infrastructure (PKI, see 9.2.3). Private DLT designs are often used if the public should not be able to access the stored data (Bott and Milkau 2016). For example, companies can use a common ledger in Supply Chain Management to collaborate, but do not want to disclose the data to other companies not involved in the collaboration.

Owing to the differences between public and private DLT designs, the consensus mechanisms’ requirements also differ. Public DLT designs must be highly scalable, especially regarding consensus finding between nodes. Consensus mechanisms for private ledgers are predominantly designed for a comparably small number of nodes to reach consensus. Most consensus mechanisms applied to private DLT design are therefore not suitable for public DLT designs. However, public consensus mechanisms can be applied to private distributed ledgers, but this comes at the cost of

efficiency (e.g., a low number of validated transactions per second or high inefficiency). For example, the blockchain concept, which is instantiated in DLT designs such as Bitcoin and Ethereum, requires each node to maintain a full replication of the data stored on the distributed ledger.

Besides the choice of using a public or private distributed ledger, consensus finding or transaction validation can be delegated to a subset of a distributed ledger's nodes (Xu et al. 2017; Yeow et al. 2018). If consensus finding is delegated to a subset of nodes (which is usually small), the DLT design is designated as *permissioned*. Since only selected nodes can validate new transactions or participate in consensus finding, fast consensus finding can be applied, which enables a throughput of multiple thousands of transactions per second (Castro and Liskov 1999). Owing to the small number of nodes involved in consensus finding, they can reach finality, which means that all of a distributed ledger's permitted nodes come to an agreement regarding the distributed ledger's current state.

In *permissionless* DLT designs, the nodes' identity does not have to be known (Yeow et al. 2018), because all of them have the same permissions. In permissionless DLT designs with a large number of nodes (e.g., Bitcoin, Ethereum), consensus finding is usually probabilistic and does not provide total finality, because it is impossible to reach finality in networks that allow nodes to arbitrarily join or leave. Consequently, the consistency between all the nodes of a public, permissionless distributed ledger can, at a certain point in time, only be assumed with a certain probability. Furthermore, a transaction appended to a distributed ledger is only assumed to be immutably stored to a certain probability. In blockchains, this probability of a particular transaction's immutability increases when new blocks are added to the blockchain (Nakamoto 2008). The period until an added transaction is considered immutable is also called confirmation latency.

Since DLT incorporates multiple computer science techniques, such as peer-to-peer networking, public key infrastructure, public key cryptography, and hashing, the technical foundations are explained in the following section for a better understanding of chapter 9.3 on the functioning of the DLT blockchain concept.

9.2.1 Hash Functions

Hash functions serve a crucial role in several systems, such as distributed ledgers, because they enable the use of digital signatures (see 9.2.3) and the verifying of data integrity. Hash functions are injective functions used to map data blocks of an arbitrary size to data of a defined, fixed size (e.g., 160 bit), also called hash values. All data blocks should ideally be transformed into an unambiguous hash value. Furthermore, hash functions must be deterministic, which is why those that a hash function produces never differ in respect of a specific input. Ideally, each possible hash value's probabilities are uniformly distributed. Hash functions should not reveal the original data, making it difficult for an attacker to guess a data block d for a hash value h such that $h = \text{hash}(d)$. Depending on how difficult it is to find a

data block d producing the targeted hash value h , it is almost impossible to reconstruct the original data block d by hashing arbitrary messages by means of a brute force search. The harder it is to reconstruct the original message, the more secure the hashed data are. Reconstructing a message from its hash value is called a preimage attack. It should be difficult to find two different m_1, m_2 messages with identical hash values $\text{hash}(m_1) = \text{hash}(m_2)$. If it is not computationally feasible to find such a second data block m_2 , the hash function is second preimage resistant. The presence of two different m_1, m_2 messages with equal hash values is called a collision. High collision resistance describes a low probability of collisions.

The unambiguity of hash values allows for verifying data integrity by storing a document's hash. If the original data block is not altered, the document's hash value will match the stored hash value. Hash values are used to verify data integrity in, for instance, the Bitcoin blockchain, where the order of the stored blocks is secured by means of the hash values (Nakamoto 2008). Hash values are also essential to create digital signatures, which are used to verify digital messages or documents' authenticity, for example, in message authentication codes (ISO/IEC 2011). In cryptocurrencies, digital signatures are commonly used (see 9.2.3) to prove ownership of a certain amount of coins (Nakamoto 2008).

Multiple secure hash algorithms (SHA), such as SHA-1, SHA-2, and SHA-3, have already been developed. Each SHA makes use of different hash functions to produce collision-resistant hash values and has a different level of security. For example, SHA-1 is less collision resistant than SHA-2 and should, therefore, be replaced by SHA-2. SHA-2 forms a family of hash algorithms that produces hash values of different lengths, such as 224 or 512 bits. Accordingly, SHA are labeled SHA-224, SHA-256, SHA-384, SHA-512, and so on. The most commonly used SHA-2 hash algorithms are SHA-256 and SHA-512. SHA-2 is widely used in security applications and protocols, including TLS and SSL. Fig. 9.2 illustrates what SHA-224 hash values look like.

```
input1 = "distributed ledger technology"
SHA256(input1) = 0x2a96ccbeaeee5fe993dce9c4b6c8922687f8df23f0d57526
                 381353b102c4ccd3

input2 = "Distributed Ledger Technology"
SHA256(input2) = 0x74178260022e2a5117ed1422e09f82c9e794eddf0cb3bb2
                 f380f10db3d60b15e
```

Fig. 9.2 Examples of SHA-256 Hash Values

9.2.2 Merkle Tree

Similar to the previously explained hash function use case, Merkle trees can be used for the efficient verification of data blocks' integrity. A Merkle tree is a data structure based on hashes and introduced by Ralph Merkle (Merkle 1988). Merkle trees are

similar in structure to an upturned tree, comprising nodes connected by vertexes. There are two types of nodes: leaf nodes and non-leaf nodes. Generally, each non-leaf node n_p is connected to two distinct childnodes n_{C1} and n_{C2} , where n_{C1} and n_{C2} can be leaf nodes or non-leaf nodes. The relevant parentnode n_p is labeled with the hash value h_p , which is the hashed concatenation of its childrens' hash values h_{C1}, h_{C2} , namely $h_p = hash(h_{C1} + h_{C2})$. At the bottom of the Merkle tree, each leaf node is a hash value of a dataset. Leaf nodes are not connected to childnodes. The nodes' hash values are therefore iteratively concatenated and hashed, beginning with the leaf nodes and continuing to the root of the Merkle tree, which is called the root hash. The root hash can even be used to verify the integrity of multiple data blocks and large data structures, which are represented by the single hash values that the leaf nodes require. Currently Merkle trees are main used in peer-to-peer networks, such as Bitcoin and Tor Network.

The data structure's one advantage is that it is not necessary to know the entire Merkle tree to verify a data block's integrity. In Fig. 9.3, for example, the integrity of data block DB2 can be verified immediately – if the tree already contains $h1$ and $h6$ – by hashing the data block and iteratively combining the result with hash $h1$ and then $h6$, and by finally comparing the result with the hash root. Terminal devices whose storage size is constrained can therefore verify data blocks' integrity without keeping a full replication of the ledger.

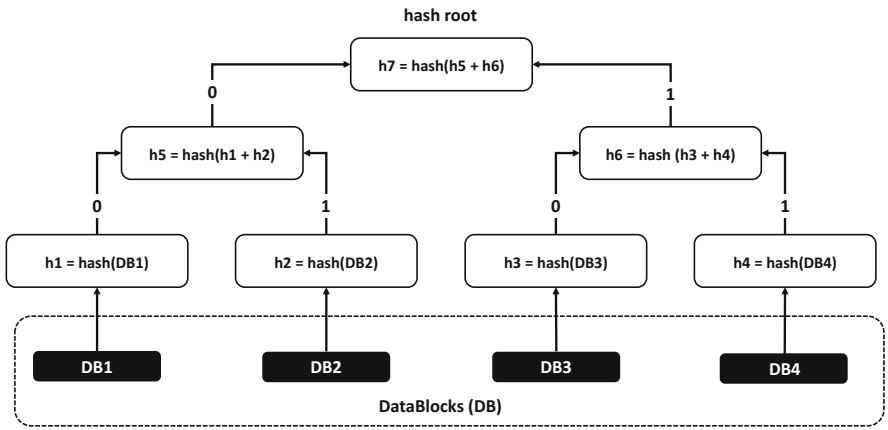


Fig. 9.3 Example of a Binary Merkle Tree

9.2.3 Public Key Infrastructure

A public key infrastructure (PKI) comprises hardware, software, policies, procedures, and roles that are used for the secure electronic transfer of data by means of an insecure network such as the Internet. A PKI manages the creation, distribution, and

revocation of digital certificates, which the use of public key cryptography requires. A public key cryptography (or asymmetric-key cryptography) uses a public and a private key. Such a public-private key pair allows identities in the network to be verified by the relevant public key, which enables the encryption and decryption of data. Furthermore, public key encryption allows the digital signing of data, which fulfills the same purpose as real signatures on paper documents and, for example, verifies authenticity when granting permission for something. An exemplary use of a PKI is to secure network connections in Secure Sockets Layer (SSL) and Hypertext Transfer Protocol Secure Sockets (HTTPS). In these uses, a PKI is applied to encrypt data exchange between entities (e.g., a browser and a server) to, for example, keep passwords secret.

A PKI is usually comprised of the following roles: certification authority (CA), registration authority (RA), subscribers, repositories, and relying parties. A CA binds public keys to subscribers by means of digital certificates. To prove the digital certificates' validity, the CA signs the issued digital certificates digitally by using a private key. The CA also issues a self-signed certificate that makes its public key available to the network participants. The CA's public key is mandatory to verify all the digital certificates that it has signed digitally, because digital signatures can only be verified by using the relevant public key. The RA is an optional, subordinate CA. The relevant CA certifies the RA, who is only permitted to issue digital certificates for specific uses. Repositories serve as a database of digital certificates. Repositories save requests for digital certificates, as well as already issued and revoked digital certificates. If a CA finds a certificate invalid, the digital certificate can be added to the repository's certification revocation lists (CRL). The CRL serves as a black list of digital certificates. The relying parties (e.g., users or terminal devices) make use of the PKI to apply public key cryptography. These parties must rely on the correctness of the digital certificates issued. The relying parties retrieve the required digital certificate from the repository to encrypt data and to verify digitally signed data. After a relying party has received a digital certificate, it is stored locally in a certification store. Once obtained, certificates do not have to be reloaded from the repository before they can be used.

As mentioned before, public key cryptography can be used to encrypt data before sending these by means of the network. The data are encrypted using the data receiver's public key, which is first retrieved from the PKI's repository. The encrypted data can only be decrypted by using the private key matching the public key used for encryption.

$$\begin{aligned} \text{encrypt}(\text{dataBlock}, \text{publicKey}_{\text{receiver}}) &\rightarrow \{\text{encryptedDataBlock}\} \\ \text{decrypt}(\text{encryptedDataBlock}, \text{privateKey}_{\text{receiver}}) &\rightarrow \{\text{dataBlock}\} \end{aligned}$$

Finding a matching private key to a corresponding public key by means of brute force is not computationally feasible (excluding quantum computers). Brute forcing tests all combinations of possible private keys systematically to find the corresponding

private key to a known public key. Once encrypted, only those who know the private key can therefore decrypt the data.

In DLT, public key cryptography is an essential approach, which is used, for example, for the creation and digitally signing of transactions. Current cryptocurrencies rely on authentication by means of digital signatures to secure users' funds. Authentication is undertaken as a part of the transaction validation process and is referred to as *proof of ownership* (Nakamoto 2008). The private key is used to digitally sign a transaction to ensure that the origin of the transaction is legitimate. Given a data block that needs to be digitally signed and the relevant private key, the following function produces a unique digital signature for that data block:

$$\text{sign}(\text{dataBlock}, \text{privateKey}) \rightarrow \{\text{signature}\}$$

To verify whether a user had signed a digitally signed block, the data block, the signature, and the public key must be known. The following function returns a binary response if a user whose public key is known signed the data block:

$$\text{verify}(\text{message}, \text{publicKey}, \text{signature}) \rightarrow \{\text{true}, \text{false}\}$$

9.2.4 Consensus Mechanisms in Distributed Ledger Technology

In the context of DLT, consensus mechanisms describe the process in which a distributed ledger's benign nodes agree on the maintained ledger replication's specific state. The state of a ledger replication within a distributed ledger can be regarded as a snapshot of the data stored in the ledger replication. Such a state can (only) be changed by adding new data to the ledger replication.

Probabilistic consensus mechanisms are better suitable for distributed ledgers that a large network maintains, because the majority of nodes must participate in consensus finding until a state can be assumed to be consistent to a certain probability. In contrast, consensus mechanisms that provide finality take all the nodes into account that are allowed to participate in the decision on what the current state of the distributed ledger should look like. After all the permitted nodes have agreed on the state, they are said to have reached consensus. Finality regarding the distributed ledgers of a small network can therefore be reached if its size and the resulting efforts for communication and synchronization is small. In the following, the most common consensus mechanisms for different types of DLT designs are briefly introduced.

Proof-of-Work

As discussed in chapter 9.1.2, PoW was originally designed to deter spam e-mails and DoS-attacks by requiring clients to do a certain amount of work before

requesting a service. In various public, permissionless DLT designs (e.g., Bitcoin), the PoW concept was extended and ultimately applied to a consensus mechanism (Nakamoto 2008). In DLT designs that use PoW as a consensus mechanism, each node must solve a computationally difficult challenge before new transactions can be included in the ledger. A reward is given to the node that first solves the particular challenge and contributes computational power. This reward takes the form of coins (e.g., BTC), with protocol defining the number of coins. In DLT designs that use blocks, the generation of blocks for a reward is called mining. Nodes that participate in the mining process are called miners. In Bitcoin, the PoW challenge, which must be solved before a node can publish a new block, is to guess an arbitrary nonce (a random string), whose hash value has at least the number of preceding zeros that a target has defined. The target is dynamically adapted to the overall computing power of a distributed ledger's nodes. If the overall computing power increases, the required number of preceding zeros in the target is increased. Consequently, the difficulty of finding the random nonce also increases. By adjusting the target's difficulty, the work to be performed as PoW is kept in keeping with the overall computing power in the distributed ledger. New blocks are therefore generated at a relatively even interval, for example, every 10 minutes in Bitcoin (Nakamoto 2008).

When applying a consensus mechanism that provides a public, permissionless DLT design with probabilistic finality (e.g., PoW), the distributed ledger can be in an inconsistent state, due to forks. A fork appears in a distributed ledger when at least two nodes issue blocks almost simultaneously, allowing the network nodes to accept different blocks as the correct successor of the last block. The consensus mechanism resolved such forks automatically by following a defined *fork resolution rule*. Bitcoin's fork resolution rule keeps the version of the distributed ledger that required the most work and replicates it on all nodes. In other words, nodes will always prefer the longest chain (Nakamoto 2008). Blocks that were mined, but not included in the distributed ledger, are called stale blocks. In Bitcoin, stale blocks are abandoned when the fork resolution rule decides another fork is the main chain.

PoW is very energy consuming, because it is computationally difficult to find a nonce with a corresponding hash value that is smaller than the target. Consequently, Bitcoin and other DLT designs based on PoW are often criticized for their inefficiency. Several alternative consensus mechanisms have been developed that, more efficiently, establish a consistent state between a distributed ledger's nodes, as described in the following.

Proof-of-Stake

Proof-of-Stake (PoS) is a less energy-consuming substitute for PoW and requires much less computational power in terms of mining. The PoS stipulates that the likelihood of a node mining the next block is closely linked to the balance of the miner's held tokens. In PoS-based cryptocurrencies, the next block's creator is chosen by means of various combinations of random selection and the stake's wealth or age. Selecting a miner according to its account balance would result in (undesirable) centralization, as the single richest node would have a permanent advantage.

Multiple selection variants were therefore developed, such as *randomized block selection*, *coin-age-based selection*, and *Delegated PoS (DPoS)*.

In *randomized block selection*, the lowest hash value, in combination with the size of the stake that a particular node holds, predicts the next block miner (Pavel 2014). Since a miner's stakes are public, each node can predict which account will next win the right to write a block, which is often criticized as weakness.

Coin-age-based selection combines randomized block selection with the "coin age" concept, a number derived from the product of the number of coins multiplied by the number of days a miner has held the coins. Nodes that own coins unspent for at least 30 days begin to compete to generate the next block. The older or larger the relevant owned coins are, the higher the probability of generating the next block. However, once a miner's stake of coins has been used to sign a block, it must start over with a "coin age" of zero and wait at least 30 days before signing another block. Furthermore, the probability of generating the next block reaches a maximum after 90 days in order to prevent very old or very large collections of stakes dominating the blockchain (King and Nadal 2012).

In *DPoS* the system uses a limited number of nodes to propose and validate blocks to be appended to the blockchain. This is meant to keep the transaction processing fast. For example, the DLT design EOS.IO⁸ uses a set of 21 randomly chosen nodes that participates in the block generation. The chosen nodes are called block producers. The distributed ledger's nodes holding coins on an EOS.IO blockchain automatically vote for block producers. The voting restarts after 126 blocks have been generated. Each node can become a block producer if enough coin holders vote for it. A node is excluded from the voting process if the node, as a block producer, has failed to produce a block and has not produced a block during the last 24 hours. A node excluded from the voting process must notify the blockchain if it wants to be considered in the voting process again (EOS.IO 2018).

The "nothing-at-stake" problem, which arises when successful block miners have nothing to lose by voting for multiple blockchain histories, thereby preventing consensus from being achieved, is an issue that can arise in PoS systems. Unlike in PoW systems, it costs little to work on multiple chains (Buterin 2014).

Practical Byzantine Fault Tolerance

Unlike the previously introduced consensus mechanisms, the practical Byzantine fault tolerance (PBFT) is used in private, permissioned DLT designs, where the network owner first vetted each of the distributed ledger's nodes. The PBFT, which enables the implementation of high-performance Byzantine fault-tolerant replicated state machines (RSMs), was introduced by Miguel Castro and Barbara Liskov in 1999 (Castro and Liskov 1999). An RSM is a system in which multiple, independent devices keep a replication of the same data set. RSM protocols allow the nodes in an RSM to function as a state machine, which receives an input in one state and generates an output based on the defined operations. The execution of an operation

⁸<https://www.eos.io/>

forms a transition from one state to a subsequent state, with a state being a view of a data set. Such RSMs can process thousands of requests per second, but have poor scalability (Castro and Liskov 1999).

To deal with the issue of Byzantine failures, the PBFT model assumes that the number of faulty nodes in the network cannot, in a given period of vulnerability, be simultaneously equal or exceed a third of the overall nodes in the distributed system. In a network that includes an assumed maximum number of faulty nodes f , the minimum number of benign nodes $\{R\}$ is therefore calculated as follows: $|\{R\}| = 3f + 1$.

Essentially, all nodes in the PBFT model are arranged in order, with one node called the primary node p , and the others backup nodes. The selection of p depends on the view number v , as follows: $p = v \bmod |R|$. v is an integer value that increments whenever the ledger data are changed, for example, after data have been added to the v . View changes are also carried out after a certain amount of time has passed without the leader node having multicasted requests.

All the nodes within the distributed ledger communicate with one another to eventually achieve consensus on the distributed ledger's next view. Each round of PBFT consensus comprises four sequential phases (Castro and Liskov 1999):

A client sends a request to p to invoke a service operation.

p multicasts⁹ the request to the backup nodes.

The backup nodes execute the request and then send a reply to the client.

The client awaits $f + 1$ replies from different nodes with the same result. This result is the operation's result.

The node requirements are that they should be deterministic and start in the same state. The operations performed on the local ledger replication determine a node's state. The result is that all the benign nodes agree on the results of an operation performed on the distributed ledger, for example, they either accept or reject the need to add data.

9.3 The Bitcoin Blockchain

Blockchain is currently one of the most popular DLT concepts. The blockchain concept was introduced with the advent of Bitcoin, which is one of the first practical solution to previously unsolved problems in distributed computing, namely the handling of nodes' fraudulent behavior (see 9.2.4) and the double-spending problem. A blockchain comprises a chronologically ordered list of blocks that are cryptographically linked to their relevant predecessor by using this previous block's hash

⁹Multicasting is a group communication in which data transmission is simultaneously directed at a group of target computers. Multicasting can be a distribution of data from one-to-many devices or from many-to-many devices.

value. A block is a data structure that stores transactions and additional data, such as a reference to the previous block (Fig. 9.4).

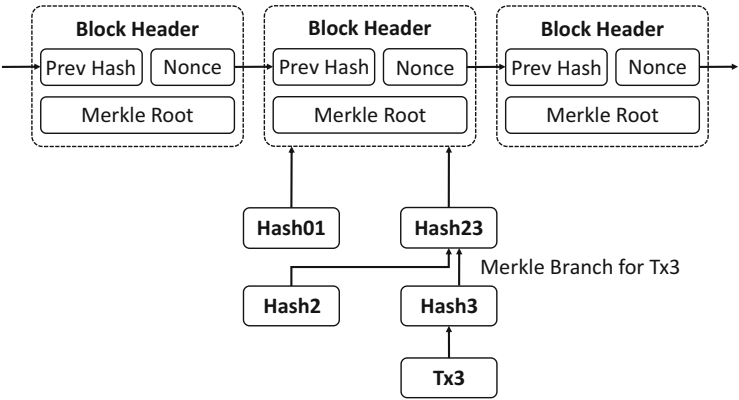


Fig. 9.4 Structure of the Bitcoin blockchain (adapted from Nakamoto (2008))

Blockchain

Blockchain is a DLT concept comprising a chain of cryptographically linked, chronologically ordered, ‘blocks’ containing batched transactions.

This chapter explains the basic functioning of the Bitcoin blockchain that inspired the development of other distributed ledgers, such as Ethereum. As of January 2019, Bitcoin had a market capitalization of more than USD 67.8 billion (CCN 2019). The schematic process of issuing and validating transactions, as well as the generation of blocks is illustrated in Fig. 9.5. Since the cryptocurrency on the Bitcoin blockchain is also called Bitcoin, the remainder of this chapter refers to the Bitcoin currency as “BTC.”

Users of Bitcoin are represented by accounts with a unique Bitcoin address, which are used to define a receiver of transactions. A Bitcoin address is a string of digits and characters that can be shared to receive BTC. Bitcoin addresses are produced by each user’s public key and consist of a string of numbers and letters. A user’s public key is first hashed. Thereafter, a 160 Bit number is generated from the hash value, using the RIPEMD160 algorithm:

$$bitcoinAddress = RIPEMD160(SHA256(publicKey))$$

End-users usually create public and private keys, storing them in a so-called wallet, and allowing them to use public key cryptography. The digital keys in a

user's wallet are completely independent of the Bitcoin protocol, and the user's wallet software can generate and manage them without reference to the blockchain or access to the Internet. Additionally, various wallets store a set of unspent transaction outputs UTXOs to proceed faster with new transactions linked to this set. Other wallets retrieve a user's unspent transaction outputs from an API that the DLT design provides. While miners store a full replication of the distributed ledger, wallets are predominantly light nodes that only store transactions related to the relevant user. Since ownership of BTC can only be proved by using public keys or users' digital signatures, users must not lose their keys otherwise the relevant BTCs cannot be accessed.

Bitcoin is based on a public peer-to-peer network, with each node maintaining a list of a few other Bitcoin nodes that it discovers during the start-up of the peer-to-peer protocol. These known nodes are called neighbored nodes. To notify each Bitcoin node of a new transaction or block, a gossip protocol is applied, which works as follows: After a Bitcoin node has received a network message, the message is multicasted to the neighbored nodes and finally propagated throughout the entire network, which is only loosely coupled, and the number of nodes that may join or leave the network is arbitrary. Consequently, Bitcoin does not have a fixed network and nodes must update their list of neighbored nodes periodically to assure messages are reliably propagated throughout the entire network.

When users initiate a new transaction, their wallets send the transaction to the distributed ledger (1). When a node receives a new transaction, it validates the transaction (2). The transaction validation includes a proof of ownership by means of digital signatures, and proof that there is sufficient balance in the user's account. Valid transactions are added to the node's local Mem-Pool, which is a temporary storage for validated transactions that will be included in future blocks, and are also multicasted to neighbored nodes. In parallel, each node in the Bitcoin blockchain performs a PoW by randomly generating arbitrary nonces, which are then hashed. The produced hash value is then compared to the current target (see 9.2.4). If the arbitrary nonce is smaller than the target, the node adds the nonce to the current block that it mines. The preceding block (the last block included in the local ledger replication) is hashed, and the produced hash value is also included in the new block. The hash value of the preceding block is used as a reference to a block's predecessor and proves the preceding block's integrity (and that of the included transactions). Furthermore, transactions are taken from the Mem-Pool, organized in a Merkle tree (see 9.2.2), and included in the new mined block (3). Subsequently, the node appends the new block to the local ledger replication and multicasts the new block to its neighbored nodes. When a neighbored node receives the new block, it validates the nonce first. If successful, the node also appends the new block to the local ledger replication (4). All transactions included in the stored block's Merkle tree are then removed from the relevant node's local Mem-Pool.

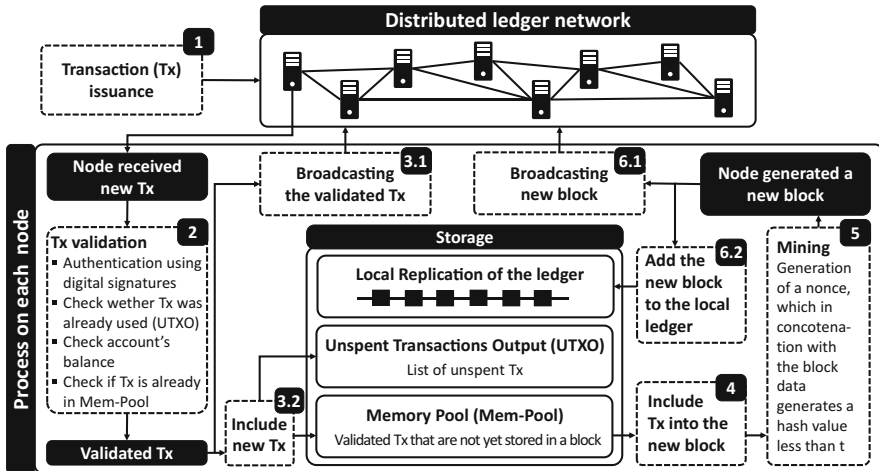


Fig. 9.5 Schematic Overview of the Inclusion of a Transaction in the Blockchain

Bitcoin nodes receive incentives for participating in the consensus mechanism; more precisely, for mining the next block (see 9.2.4). A node receives a reward in the form of BTC coins, with a mathematical function determining the amount, for completing this task. This function, also called reward halving, halves Bitcoin block-mining rewards every 210,000 blocks. In this sense, the received benefit per block is reduced due to the growing number of mined blocks. For example, the reward decreased from 12.5 BTC to 6.25 BTC on May 25, 2020. In other words, it takes more effort to obtain the same amount of BTC tokens over time. Once mined, BTC tokens can be traded on a dedicated market place with other cryptocurrencies or exchanged for fiat money. In addition to the block reward, miners receive a transaction fee for each transaction included in the mined block. Miners can prioritize transactions with high transaction fees rather than those with low transaction fees when including transactions in blocks. To increase the transaction fee, users can define what they are willing to pay when issuing a transaction through their wallet. After the total amount of BTC has been reached (BTC 21,000,000), miners will only be incentivized to keep on sharing their computational power if they receive transaction fees as a reward when validating a transaction and including it into a block.

The transfer of BTC from one Bitcoin account to another is represented in transactions' directed graph (DAG) (Fig. 9.6). Each transaction has at least one input representing a debit against the transaction originator's Bitcoin account. Transactions need at least one output (i.e., payouts added to a Bitcoin account). A new transaction's inputs refer to previously created transactions with no successive transactions as yet. Consequently, these transactions' outputs are not used and are called unspent transaction outputs (UTXOs). Any new transaction provides a new UTXO that can be used to transfer BTC in subsequent transactions. Each UTXO

contains a locking script or *scriptPubKey*. Locking scripts define the condition (e.g., a corresponding signature) that must be fulfilled to spend a UTXO. Transaction inputs refer to previous UTXOs and contain an unlocking script or *scriptSig*. Unlocking scripts require defining the conditions in the locking script before a transaction can be performed. A user account can have multiple UTXOs, which can be combined to transfer BTC. To transfer BTC to a Bitcoin account, the originator of a transaction must digitally sign the hash(es) of the previous transaction(s) and the receiver’s public key. The produced digital signature is then added to the end of the new transaction (see Fig. 9.6), with the previous transaction(s) serving as input(s). The BTC receiver can verify the signature in order to verify the chain of ownership. Any node can verify the proof of ownership when validating a new transaction. If the transaction is invalid, the node will reject it and synchronously return a rejection message to the originator. All BTC locked in a transaction are transferred when this transaction is unlocked. If the amount of BTC locked in a transaction does not match the amount of BTC that should be transferred, the transaction’s originator receives the remaining BTC in an additional transaction, which is linked to a second transaction output. The change equals the difference between the sum of the BTC unlocked in the relevant transactions, the amount of spent BTC, and additional transactions fees.

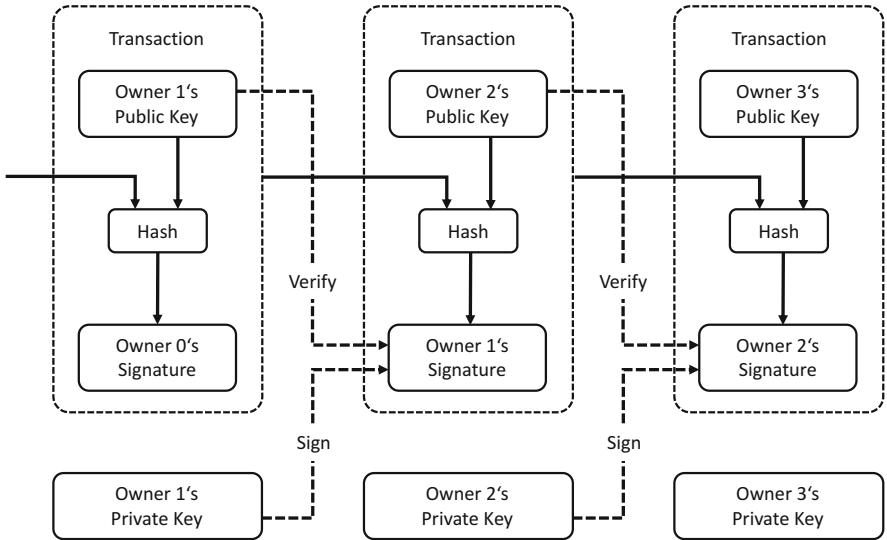


Fig. 9.6 Bitcoin Transaction Chain (adapted from Nakamoto (2008))

Bitcoin has several constraints, some of which have already been addressed during the last few years but have not been satisfactory solved. For instance, Bitcoin’s throughput is about 7 transactions per second (tps) and is constrained by its maximum block size of 1 MB (Croman et al. 2016). Owing to this relatively poor throughput of Bitcoin, a transaction may be delayed, depending on the number of

uncommitted transactions that remains in the Mem-Pool. Consequently, advanced blockchain implementations approaches have been developed, such as Ethereum, HyperLedger Fabric¹⁰, and ZCash¹¹. These projects introduced several improvements, such as the higher throughput (e.g., HyperLedger Farbic with about 3,500 tps), and the unidentifiability and untraceability in ZCash.

Nevertheless, the improvement of a particular DLT characteristic usually comes at the cost of other DLT characteristics. For example, practical Byzantine fault-tolerance (see 9.2.4) can archive a much higher throughput than PoW, but requires a small number of nodes to participate in the consensus mechanism, which decreases its resistance to individual nodes' malicious behavior (Kannengießer et al. 2019). Consequently, there are many specialized DLT designs that fulfill the requirements for a particular use case.

Although blockchains are often considered immutable, public, permissionless blockchains are especially vulnerable to attacks on the stored data's integrity (Eyal and Sirer 2014; Deirmentzoglou et al. 2019). Owing to the occurrence of forks (see 9.2.4), the integrity of blockchains can be specifically deterred in, for example, 51% of attacks. In 51% attacks, more than 50% of the nodes are assumed to have a malicious intent. In this case, the distributed ledger's transaction history should be reversed, and the stored data's integrity is no longer assured. Other attacks, such as selfish-mining (Göbel et al. 2016) and the balance attack (Natoli and Gramoli 2017), can also pose a threat to the distributed ledger's integrity. However, these attacks lie outside this section's scope.

9.4 Smart Contracts

When renting an apartment, a deposit account is often set up to allow the tenant to pay the rent sum into it. The tenant cannot withdraw money from this account without the landlord's consent. Likewise, the landlord cannot withdraw the money without the tenant's consent. Only if both parties agree, can the money be withdrawn. Banks usually offer and maintain such accounts. However, DLT now enables the digital mapping of such deposit accounts, as well as even more complex agreements without a third party, such as a bank, by means of smart contracts. Smart contracts are computer programs in which a business logic is formalized; they therefore allow secure transaction issuance without the need for third parties. The transactions that a smart contract issues, have similar characteristics to those that users issue: they are traceable and immutable. The purpose of smart contracts is to reduce the transaction costs associated with contracting and to accelerate the enforcement of agreements formalized in smart contracts. Nick Szabo (Szabo 1997) coined the term smart contract.

¹⁰<https://www.hyperledger.org/projects/fabric>

¹¹<https://z.cash/>

Bitcoin already provides the option to determine certain conditions that need to be fulfilled before a transaction can be issued, which means that, in principle, it can be used to deploy smart contracts. In order for Bitcoin to do so, its scripting language consists of a predefined functionality, the Bitcoin OP_CODE, which is a *Turing-incomplete* script language enabling the use of simple forms of smart contracts, such as multi-signature accounts, time locks, and atomic cross-chain trading, without requiring a trusted third party (Atzei et al. 2018).

The Ethereum blockchain is the most popular example of smart contracts' use. The Ethereum blockchain comes with a *Turing-complete* programming language (Solidity) and a virtual machine, the Ethereum virtual machine (EVM), which serves as an environment for the execution of smart contracts. EVM enables the execution of a custom (sophisticated) logic formalized in smart contracts for automate processes, such as withdrawals of tokens (Buterin 2018). Ethereum was the first DLT design to provide a Turing-complete environment for the execution of smart contracts, which can be developed by using an HLL (Buterin 2018). To deploy smart contracts on the Ethereum blockchain, Ethereum makes use of tokens that follow a defined structure, namely the ERC-20 token standard¹². In Ethereum, smart contracts are first compiled to executable byte code and thereafter included in a transaction according to the ERC-20 standard. The transaction is issued to the Ethereum blockchain and regularly validated by each node. After the transaction has been included in the blockchain, the issuer receives the smart contract's address, which is similar to an account's address and is used to interact with the deployed smart contract. The smart contract will always retain this address and is stored on each of the Ethereum blockchain's nodes, as it is in financial transactions. Likewise, after a smart contract's deployment, it can no longer be maintained or deleted. In order to execute a deployed smart contract, a transaction must be sent to this smart contract's address. Like any other transaction, triggering a smart contract is propagated throughout the network, with each node executing the relevant smart contract asynchronously.

Smart contracts are not restricted to data stored on the same distributed ledger (on-chain), but can also communicate with external (off-chain) services, such as an external REST API. External services used as data feeds and called from smart contracts are called oracles.

9.5 Applications of Distributed Ledger Technology

As discussed in the previous sections, DLT is steadily becoming more versatile in terms of applicable use cases. The following examples introduce a selection of highly discussed applications that go beyond DLT's application in cryptocurrencies. Following the understanding of DLT applications in the previous section, all the following applications can also be considered examples of DLT applications.

¹²https://theethereum.wiki/w/index.php/ERC20_Token_Standard

9.5.1 *Financial Technology*

Payments

Cryptocurrency coins (e.g., Bitcoin, Ether, or ZCash) were developed to facilitate online payments without the need for a trusted third party, such as financial service providers like banks. No central bank or public authority issues cryptographic coins and are therefore not usually linked to a legally defined currency (fiat currency). However, certain natural or legal persons do accept cryptocurrencies as a means of exchange. Applications to use cryptocurrencies in the financial industry can be found in areas such as payment transactions, securities settlement, and trade financing. DLT is promising as it could accelerate all of these areas by means of automation and increasing transparency while decreasing transaction costs (WTO 2018).

To simplify the settlements of trade and foreign exchange transactions with different currencies, banks often open a reference account (nostro account) in another bank to provide the targeted currency. For example, if a Ghanaian company buys or sells products in Germany often, but does not have a physical presence there, this company will ask its local bank to set up a Euro account. The Ghanaian bank will then look for a bank in Germany that offers Euro and ask it to open a bank account. The Euro bank will set up a bank account that is not a typical checking account. In this constellation, both banks need to keep records of the amount of money that one bank keeps on behalf of the other. When DLT is used for the accounting of digital assets, it can become more efficient and reliable (Brown et al. 2016). Owing to the decreased bookkeeping effort, DLT can provide both account holders and their service providers with real-time transparency regarding the available and forecasted liquidity in the nostro account. In the long-term, banks might not even look for a partner bank to open a new account, because payments might be transferred directly to the targeted bank account without considering the bank's current country. Consequently, DLT has the potential to reduce the number of required intermediaries.

The Interbank Information Network (IIN) was found by J.P. Morgan to carry out international payment transactions based on the private DLT design called Quorum¹³. In 2018, more than 75 banks participated in the IIN (Morgan 2018).

Seed Funding

To finance start-ups, a new form of financing based on DLT was developed in 2015 – the initial coin offerings (ICOs). ICOs represent a process through which companies or other project sponsors raise capital for their projects in exchange for tokens. ICOs are used for the early-stage financing of companies or projects that often have only one project concept, like fundraising. ICOs differ from classic early-stage financing by means of a venture capital (VC) company in that private investors can support them directly. Usually, several intermediaries, such as investment banks, notaries, or crowd-funding platforms (e.g., Kickstarter), are

¹³<https://www.jpmorgan.com/global/Quorum>

involved in fundraising, which impedes the fundraising process. These intermediaries can be (partially) replaced by using DLT for the fundraising, because it allows small-sized companies and start-ups to raise funds for projects at very little costs. A recent study shows that the funds raised via ICOs have nearly quadrupled from USD 4.1 billion in 2017 to USD 15.5 billion in early 2018 (Ernst and Young Global 2018). However, 86 percent of the tokens issued in 2017 are traded below their initial price. ICOs from 2017 show an average loss of 66 percent compared to their peak value. Overall, the volumes achieved indicate that, basically, ICOs might be an attractive form of financing for start-ups, although the extent to which they are suitable as a viable form of investment remains to be seen in view of the described losses.

9.5.2 Health Care

Currently, people have to manually reconcile medical health records across, for example, laboratories, physicians, and pharmacies. Arguably, this requires people to maintain a list of all stakeholders that process their data. For instance, it may not be clear what kind of drugs a patient is currently taking. Although data standards are better than ever, each electronic health record (EHR) stores data by using different workflows, making it unclear which person recorded what data and at which point in time. DLT is a promising means of overcoming these issues. MedRec is a decentralized record management system that handles EHRs by means of blockchain technology (Azaria et al. 2016) and aims to reduce the efforts required to obtain a full representation of a particular patient's EHR to ensure the best possible treatment.

MedRec stores the EHR's digital signature on a blockchain and patients are ultimately in control of who views their data. The signature assures that an unaltered copy of the EHR is obtained. MedRec also shifts the focus of control from the institution (e.g., a hospital) to the patient, and, in turn, allows patients to take charge of their EHR management, which they may, however, find a burden. MedRec is therefore faced with building an interface that patients will find easy to use.

9.5.3 Supply Chain Management

Escrow Service in Supply Chain Management

The use of DLT addresses many supply chain management issues. Consider the following example: When a company A buys a product from company B, an agreement must be made to determine payment, delivery, and the product conditions. There are basically two options for managing the payment: first, B delivers the product to A without prior payment, or, second, A first pays for the product without knowing whether it meets the requirements. To solve this issue, A and B usually

utilize a financial service that holds the money for the product in trust. Such financial services require the presence of, among others, a financial institution and a notary, which is why they are costly and time-consuming. Smart contracts can revolutionize financial services in terms of their cost and speed (BMO et al. 2018). A and B can, however, formalize the conditions for payment in a smart contract. Company A then sends the requested funds to the smart contract. Neither A nor B can access the sent funds as long as the conditions defined in the smart contract are not fulfilled; that is, A must first confirm that the received product meets its requirements. After the product has arrived at A, it can check this product and either confirm the payment in the smart contract or negotiate with B if the product has defects.

Product Tracking via Digital Twins

During the last few years, the public have become very interested in sustainability and economic awareness, demanding more transparency in supply chains. For example, end-customers, companies, and governmental institutions are interested in tracking products' provenance in order to determine their production conditions. End-customers want to know where a product (e.g., food or clothes) comes from. Consequently, some companies now add codes to their products, allowing their origin to be traced online. By addressing these information needs, companies can foster their image in terms of, for instance, their ecological sustainability. Imported products' origin and production conditions are of special importance for governmental institutions. For example, reliable provenance tracking allows blood diamonds, which violate diamond mining's ethical standards, to be identified. However, stored data used for provenance tracking are currently relatively easy to manipulate, because they are either based on paper documents or stored in a central database. DLT is a promising way of overcoming the issue of post hoc modifiable documentation. This concept involves creating a digital representation of a real-world product, a digital twin. Assigning a digital twin unambiguously to a real-world entity (e.g., a product) requires the generation of a unique identifier (UID) that only the producer of the relevant product can generate. Such a UID can be based on, for example, a product's inherent characteristics (e.g., surface texture), a specific color, or a specialized hardware device (e.g., a near-field communication device). The product's UID is then registered on the distributed ledger, which also allows all the relevant data from current paper documents to be added. Each time the product is handed to another individual or organization, a transaction on the DLT design documents the change of ownership.

Everledger¹⁴ is a successful tracker of diamonds' provenance, aimed at protecting diamonds' value throughout the supply chain and facilitating responsible and ethical sourcing of diamonds. The Everledger system allows blood diamonds to be identified, the reliable retrieval of their characteristics, such as their carat and size, and the approval of diamond certifications. When a diamond is registered in the system, a laser gives it a unique inscription, which represents its UID and is used as a reference

¹⁴<https://www.everledger.io/>

to its digital twin on the blockchain. To check a diamond's provenance, the relevant UID can be looked up on the distributed ledger. Currently, more than 1.6 million diamonds are registered on a blockchain.

End-customers' demand for transparency regarding the production and delivery of products in the food sector has increased over the last few years. Walmart and IBM therefore jointly developed a DLT-based system for tracing the origin of food products, focusing on lettuce initially (Corkery and Popper 2018). In the meantime, there are plans to adapt food safety via blockchain to the provenance tracking of cereals (Pyrzenski 2017), coffee (Widdifield 2018), and fishing (Lehikoinen 2018).

Summary

DLT is a promising technology that enables the realization of distributed ledgers, which are a type of distributed database with new data appended to the end of the ledger. Generally, distributed ledgers only allow read and create operations and only append new data to the ledger. Distributed ledgers can be applied in settings requiring a highly available, immutable, and reliable database as a shared data storage for different actors who do not trust one another.

Originating from the idea of a completely decentralized cryptocurrency that does not require the presence of banks or governments, the idea of digital money arose in the 1990s. However, the first digital currencies, like eCash, still required intermediaries to administer digital money and to prevent double-spending. Double-spending refers to the problem of spending the same digital asset at the same time for multiple purposes. Further, subsequent cryptocurrencies, such as e-gold, could not assert themselves against, for example, credit cards. Satoshi Nakamoto's whitepaper on Bitcoin, published in 2008, introduced the first fully decentralized cryptocurrency, which became popular worldwide in the following years. Bitcoin is the DLT generation 1.0, as it first allowed the exchange of values based on self-generated coins without the need for an intermediary. DLT generation 1.0 was developed further to make it better suited for purposes other than cryptocurrencies. Technical advancements were made, which not only improved DLT's performance, but also its flexibility regarding application areas. Consequently, DLT is now used in industrial applications. The types of DLT designs can be divided into private and public, and these in turn into permissioned and permissionless designs. While private and permissioned DLT designs promise a high throughput, public permissionless designs have a higher degree of anonymity and decentralization.

Bitcoin and distributed ledgers developed later, such as Ethereum, combine technologies from cryptography, distributed systems, and game theory. These DLTs' nodes are interconnected in a peer-to-peer network and achieve a consistent state by means of a consensus mechanism that is Byzantine fault tolerant. A consistent state means that all of the distributed ledger's nodes agree on the stored replication ledger having the same content and structure. Byzantine fault tolerance describes a system's ability to reach a consistent state, although the nodes may (temporarily) not be available or could even be malicious.

The essential cryptographic concepts used in DLT designs are hash functions and public key cryptography. DLT designs use hash functions to maintain data integrity. Public key cryptography proves the ownership of assets locked in transactions. The public key cryptography requires a public key and a corresponding private key, with data being encrypted by using the public key. The corresponding private key is used to decrypt encrypted data. Furthermore, data can be digitally signed by using the private key. The signing of data is essential for DLT, as this makes the ownership of digital assets traceable. Among other things, hash functions are used for Merkle trees to maintain stored transactions' integrity and to quickly check stored data's correctness.

The description of the technical basics was completed by presenting three predominantly used consensus mechanisms: proof-of-work (PoW), proof-of-stake (PoS), and practical Byzantine fault tolerance (PBFT). PoW is used in the Bitcoin protocol and is a crucial part of the first consensus mechanisms applied to DLT. The more energy efficient PoS overcomes PoW's intense energy consumption and takes the miners' particular stakes into consideration. The PBFT approach is frequently used for permissioned DLT designs. Based on the technical basics, a description was subsequently provided of the Bitcoin blockchain's basic functionality.

Smart contracts are programs that can be deployed on a distributed ledger; they allow agreements to be formalized and to be reliably enforced on the distributed ledger. When a transaction is sent to a smart contract, the latter is executed. Each node of the distributed ledger executes the smart contract independently. Subsequently, consensus is sought regarding the smart contract's results. It is often necessary to use external data feeds (oracles) to obtain data from the real world in order to achieve consensus.

DLT is of interest in domains such as finance, healthcare, and supply chain management. Several prototypes have already been developed, but only a few are actually used in industries.

Questions

1. What was the motivation behind the development of a crypto currency?
2. What is DLT and how can blockchain be classified within DLT?
3. What are inherent characteristics of DLT?
4. What is a digital signature and what purpose does it serve in DLT?
5. What are hash functions used for in DLT and why?
6. How does the Bitcoin blockchain work?
7. What are smart contracts and what can they be used for?

References

- Abbasi AG, Khan Z (2017) VeidBlock: verifiable identity using blockchain and ledger in a software defined network. Paper presented at the 10th international conference on utility and cloud computing, Austin, TX, 5–8 Dec 2017

- Alansari S, Paci F, Margheri A, Sassone V (2017) Privacy-preserving access control in cloud federations. Paper presented at the 10th IEEE international conference on cloud computing, Honolulu, HI, 25–30 June 2017
- Atzei N, Bartoletti M, Cimoli T, Lande S, Zunino R (2018) SoK: unraveling bitcoin smart contracts. Paper presented at the international conference on principles of security and trust, Thessaloniki, 16–19 Apr 2018
- Azaria A, Ekblaw A, Vieira T, Lippman A (2016) MedRec: using blockchain for medical data access and permission management. Paper presented at the 2nd international conference on open and big data, Vienna, 22–24 Aug 2016
- Back A (1997) A partial hash collision based postage scheme. <http://www.hashcash.org/papers/announce.txt>. Accessed 23 Sept 2019
- BMO, CaixaBank, commerzbank, Group E, IBM, UBS (2018) First pilot client transactions successfully executed on Batavia global trade finance platform. https://web.archive.org/web/20190313101647/https://www.commerzbank.com/media/presse/archiv_1/mitteilungen/2018_1/2018-04-19_PR_Batavia_First_Transactions_EN.pdf. Accessed 12 Mar 2019
- Bott J, Milkau U (2016) Towards a framework for the evaluation and design of distributed ledger technologies in banking and payments. *J Paym Strateg Syst* 2:153–171
- Brown RG, Carlyle J, Grigg I, Hearn M (2016) Corda: an introduction. https://docs.corda.net/releases/release-M7.0/_static/corda-introductory-whitepaper.pdf. Accessed 19 Sept 2019
- Buterin V (2014) On stake. <https://blog.ethereum.org/2014/07/05/stake/>. Accessed 23 Sept 2019
- Buterin V (2017) The meaning of decentralization. <https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274>. Accessed 19 Sept 2019
- Buterin V (2018) Ethereum whitepaper. <https://github.com/ethereum/wiki/wiki/White-Paper>. Accessed 19 Sept 2019
- Castro M, Liskov B (1999) Practical Byzantine fault tolerance. Paper presented at the 3rd symposium on operating systems design and implementation, New Orleans, LA, 22–25 Feb 1999
- CCN (2019) Marketcap with prices of cryptocurrencies like bitcoin and ethereum. <https://www.ccn.com/marketcap/>. Accessed 23 Sept 2019
- Chaum D (1983) Blind signatures for untraceable payments. In: Chaum D, Rivest RL, Sherman AT (eds) *Advances in cryptology*. Springer, Boston, MA, pp 199–203
- Chohan (2017) Cryptocurrencies: a brief thematic review. https://papers.ssm.com/SOL3/PAPERS.CFM?ABSTRACT_ID=3024330. Accessed 19 Sept 2019
- Corkery M, Popper N (2018) From farm to blockchain: Walmart tracks its lettuce. *New York Times*, 24 Sept 2018
- Croman K, Decker C, Eyal I, Gencer AE, Juels A, Kosba A, Miller A, Saxena P, Shi E, Gün Sirer E (2016) On scaling decentralized blockchains. Paper presented at the financial cryptography and data security, Christchurch, 3–7 Apr 2016
- Cryptography F (2005) GP4.3 – growth and fraud – case #3 – phishing. <https://www.financialcryptography.com/mt/archives/000609.html>. Accessed 19 Sept 2019
- Cynthia D, Moni N (1993) Pricing via processing, or, combatting junk mail, advances in cryptology. Paper presented at the annual international cryptology conference, Santa Barbara, CA, 22–26 Aug 1993
- Dagher G, Mohler J, Milojkovic M, Marella P (2018) Ancile: privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustain Cities Soc* 39:283–297
- Dai W (1998) B-money. <https://en.bitcoin.it/wiki/B-money>. Accessed 19 Sept 2019
- Deirmentzoglou E, Papakyriakopoulos G, Patsakis C (2019) A survey on long-range attacks for proof of stake protocols. *IEEE Access* 7:28712–28725
- e-gold (2004) e-gold statistics. <https://web.archive.org/web/20040711020115/http://www.e-gold.com/stats.html>. Accessed 23 Sept 2019
- EOS.IO (2018) EOS.IO technical white paper v2. <https://github.com/BlockchainTranslator/EOS/blob/master/TechDoc/EOS.IO-Technical-WhitePaper-v2.md>. Accessed 23 Sept 2019

- Ernst & Young Global (2018) EY study: initial coin offerings (ICOs). The class of 2017 – one year later. [https://www.ey.com/Publication/vwLUAssets/ey-study-ico-research/\\$FILE/ey-study-ico-research.pdf](https://www.ey.com/Publication/vwLUAssets/ey-study-ico-research/$FILE/ey-study-ico-research.pdf). Accessed 19 Sept 2019
- Eyal I, Sirer EG (2014) Majority is not enough: bitcoin mining is vulnerable. *Commun ACM* 61(7):95–102
- Göbel J, Keeler H, Krzesinski A, Taylor P (2016) Bitcoin blockchain dynamics: the selfish-mine strategy in the presence of propagation delay. *Perform Eval* 104:23–41
- Gold & Silver Reserve I (1999) e-gold shopping cart interface specification. https://web.archive.org/web/20000815224815/http://www.e-gold.com/docs/e-gold_SCI.pdf. Accessed 23 Sept 2019
- Hadjicostis M (2013) Bank of cyprus big savers to lose up to 60 percent. *CNBC*, 30 Mar 2013
- Hileman G, Rauchs M (2017) Global blockchain benchmarking study. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3040224. Accessed 20 Sept 2019
- Hughes SJ, Middlebrook ST, Broox WP (2007) Developments in the law concerning stored-value cards and other electronic payments products. *Bus Law* 63(1):237–269
- IOS/DIS (2018) Ergonomics of human-system interaction – part 11: Usability: definitions and concepts. <https://www.iso.org/standard/63500.html>. Accessed 20 Sept 2019
- ISO/IEC (2009) Information technology – security techniques – information security management systems – overview and vocabulary. <https://www.iso.org/standard/73906.html>. Accessed 20 Sept 2019
- ISO/IEC (2011) Information technology – security techniques – message authentication codes (MACs) – Part 1: Mechanisms using a block cipher. <https://www.iso.org/standard/50375.html>. Accessed 20 Sept 2019
- Jakobsson M, Juels A (1999) Proofs of work and bread pudding protocols. In: Preneel B (ed) *Secure information networks: communications and multimedia security*. Springer, Boston, MA, pp 258–272
- Kannengießer N, Lins S, Dehling T, Sunyaev A (2019) What does not fit can be made to fit! Trade-offs in distributed ledger technology designs. Paper presented at the 52nd Hawaii international conference on system sciences, Maui, HI, 8–11 Jan 2019
- King S, Nadal S (2012) PPCoin: peer-to-peer crypto-currency with proof-of-stake. <https://bitcoin.peryaudio.org/vendor/peercoin-paper.pdf>. Accessed 20 Sept 2019
- Kursh SR, Gold NA (2016) Adding fintech and blockchain to your curriculum. *Bus Edu Innov J* 8(2):6–12
- Lamport L, Massa M (2004) Cheap Paxos. Paper presented at the international conference on dependable systems and networks, Florence, 28 June–1 July 2004
- Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. *ACM Trans Program Lang Syst* 4(3):382–401
- Lehikoinen T (2018) This summer, fishing in Finland means food traceability on the menu. <https://www.ibm.com/blogs/blockchain/2018/07/this-summer-fishing-in-finland-means-food-traceability-on-the-menu/>. Accessed 19 Sept 2019
- Li Z, Wu H, King B, Miled ZB, Wassick J, Tazelaar J (2017) On the integration of event-based and transaction-based architectures for supply chains. Paper presented at the 37th IEEE international conference on distributed computing systems workshops, Atlanta, GA, 5–8 June 2017
- Merkle RC (1988) A digital signature based on a conventional encryption function. Paper presented at the annual international cryptology conference, Santa Barbara, CA, 16–20 Aug 1987
- Morgan JP (2018) J.P. Morgan Interbank Information NetworkSM expands to more than 75 banks. <https://www.jpmorgan.com/country/US/en/detail/1320570135560>. Accessed 19 Sept 2019
- Morris DZ (2018) Bitcoin hits a new record high, but stops short of \$20,000. *Fortune*, 17 Dec 2018
- Myerson RB (2004) *Game theory: analysis of conflict*, 6th edn. Harvard University Press, Cambridge, MA
- Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. https://s3.amazonaws.com/academia.edu.documents/54517945/Bitcoin_paper_Original_2.pdf?response-content-disposition=inline%3B%20filename%3DBitcoin_A_Peer-to-Peer_Electronic_Cash_S.pdf&X-Amz-

- Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20190920%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190920T114421Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=2da287fe23d2b16841842150c8a6f629efc062afab512ca1de042376fc568b83. Accessed 20 Sept 2019
- Natoli C, Gramoli V (2017) The balance attack or why forkable blockchains are ill-suited for consortium. Paper presented at the 47th annual IEEE/IFIP international conference on dependable systems and networks, Denver, CO, 26–29 June 2017
- Pass R, Shi E (2017) The sleepy model of consensus. Paper presented at the international conference on the theory and application of cryptology and information security, Hong Kong, 3–7 Dec 2017
- Pavel V (2014) BlackCoin's proof-of-stake protocol v2. http://bitpaper.info/serve/AMIfv96zY1Qy1kHDKj-0P5_SZMG5ffHm8EyOVwBzPTqbINP0-R3femZWkzk08i-ISg5ZgACMrDCMHH-jovVKeXoXlrSy-zF7NZt7NMWRpT-gmWDrW-Qz6NdOudmOvYlXOreooL3-YK8mf6rYFHGQR6Vn5aFwZSAm625XNYpjoCc0OuuIMzCsc.pdf. Accessed 20 Sept 2019
- Pfitzmann A, Hansen M, Köhntopp M (2006) Anonymity, unlinkability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.421&rep=rep1&type=pdf>. Accessed 20 Sept 2019
- Popov S (2018) The tangle. http://tanglereport.com/wp-content/uploads/2018/01/IOTA_Whitepaper.pdf. Accessed 19 Sept 2019
- Pyrzinski D (2017) I'll only eat blockchain cereal with a food safety label on the box. <https://www.ibm.com/blogs/blockchain/2017/09/ill-only-eat-blockchain-cereal-with-a-food-safety-label-on-the-box/>. Accessed 19 Sept 2019
- Sompolinsky Y, Lewenberg Y, Zohar A (2018) SPECTRE: serialization of proof-of-work events: confirming transactions via recursive elections. <https://medium.com/@avivzohar/the-spectre-protocol-7dbbebb707b5>. Accessed 20 Sept 2019
- Szabo N (1997) Formalizing and securing relationships on public networks. First Monday 2(9)
- Widdifield J (2018) Brewing blockchain: tracing ethically sourced coffee. <https://www.ibm.com/blogs/blockchain/2018/08/brewing-blockchain-tracing-ethically-sourced-coffee/>. Accessed 19 Sept 2019
- WTO (2018) World trade report 2018. https://www.wto.org/english/res_e/publications_e/world_trade_report18_e.pdf. Accessed 19 Sept 2019
- Xu X, Weber I, Staples M, Zhu L, Bosch J, Bass L, Pautasso C, Rimba P (2017) A taxonomy of blockchain-based systems for architecture design. Paper presented at the IEEE international conference on software architecture, Gothenburg, 3–7 Apr 2017
- Yeow K, Gani A, Ahmad RW, Rodrigues JJPC, Ko K (2018) Decentralized consensus for edge-centric internet of things: a review, taxonomy, and research issues. IEEE Access 6:1513–1524

Further Reading

- Buterin V (2018) Ethereum whitepaper. <https://github.com/ethereum/wiki/wiki/White-Paper>. Accessed 19 Sept 2019
- Glaser F, Bezzenberger L (2015) Beyond cryptocurrencies – a taxonomy of decentralized consensus systems. 23rd European conference on information systems, Münster, pp 1–18
- Göbel J, Krzesinski AE (2017) Increased block size and Bitcoin blockchain dynamics. 27th international telecommunication networks and applications conference, Auckland, 2017, pp 1–6

- Kannengießer N, Lins S, Dehling T, Sunyaev A (2019) What does not fit can be made to fit! Trade-offs in distributed ledger technology designs. Paper presented at the 52nd Hawaii international conference on system sciences, Maui, HI, 8–11 Jan 2019
- Kannengießer N, Pfister M, Greulich M, Lins S, Sunyaev A (2020) Bridges between islands: cross-chain technology for distributed ledger technology. Presented at the 53rd Hawaii international conference on system sciences, Waikoloa, Hawaii
- Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. https://s3.amazonaws.com/academia.edu.documents/54517945/Bitcoin_paper_Original_2.pdf?response-content-disposition=inline%3B%20filename%3DBitcoin_A_Peer-to-Peer_Electronic_Cash_S.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20190920%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190920T114421Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=2da287fe23d2b16841842150c8a6f629efc062afab512ca1de042376fc568b83. Accessed 20 Sept 2019
- Yeow K, Gani A, Ahmad RW, Rodrigues JJPC, Ko K (2018) Decentralized consensus for edge-centric internet of things: a review, taxonomy, and research issues. *IEEE Access* 6:1513–1524

Chapter 10

The Internet of Things



Abstract

Building on the Internet's success story over the past decades, the Internet of Things will profoundly change how people consume information and interact with their immediate environments. This chapter introduces the Internet of Things as a paradigm in which not only human-to-human and human-to-machine communication, but also machine-to-machine communication between smart everyday objects occur over the Internet. Besides a brief definition and overview of the Internet of Things' historical background in the early 1990s, several of its enabling technologies and basic concepts are also covered. Furthermore, this chapter also presents an overview of important architectural models of the Internet of Things put forward by researchers and practitioners. To conclude this introduction, this chapter deals with several common use cases of the Internet of Things, such as smart homes, smart cities, and the Industrial Internet of Things (with a specific focus on the implications for the energy and health care sectors), as well as important challenges to and potential future developments of the Internet of Things.

Learning Objectives of this Chapter

This chapter's main learning objective is to help students understand the Internet of Things concept and its implications for individuals, organizations, and society. Having read this chapter, students will understand the basics of the Internet of Things, including its essential characteristics and historical origins; students will also be able to differentiate between the Internet of Things and related concepts, such as ubiquitous computing. Students will specifically learn about the enabling technologies that contribute to the Internet of Things' realization. This chapter will also help students differentiate between the main architectural models put forward for the Internet of Things. Lastly, students will not only learn how the Internet of Things affects and changes central aspects of people's daily lives, but also which leading challenges are associated with these changes.

Structure of this Chapter

This chapter is structured as follows: The first section introduces the Internet of Things concept by providing a definition and presenting its roots, as well as its history. The second section presents the Internet of Things' enabling technologies (including tagging, sensing, smart, and miniaturization technologies), core concepts, and architectural models, followed in the third section by a description of selected application areas of the Internet of Things (i.e., smart homes, smart cities, and the Industrial Internet of Things). The chapter concludes with a discussion of certain challenges associated with the proliferation of the Internet of Things and a viewpoint on its future prospects.

10.1 Introduction of the Internet of Things

10.1.1 *Definition and Characteristics*

Technological progress is changing the way in which people use information systems at work and in their free time. Especially the Internet has profoundly changed how people consume and exchange information, and how they interact with each other. Driven by improvements in microprocessor, storage, broadband network, and sensor technologies, as well as increasingly efficient power management, more areas of everyday life are increasingly being computerized and connected to the Internet. Nowadays, Internet use is not restricted to humans accessing the Internet over classic devices, such as personal computers and mobile phones, but increasingly expands to objects of everyday life and the immediate environment (e.g., light bulbs, refrigerators, or trains) that are connected to the Internet. In 2019, for example, the number of active Internet users surpassed four billion people, which is more than half of the global population (Statista 2019a). At the same time, the number of internet-connected devices (e.g., smart phones, wearables, smart home devices) is expected to exceed 30 billion by 2020 and 75 billion by 2025 (Statista 2019b). Building on this trend of connected everyday objects and environments, the Internet of Things is a paradigm in which not only human-to-human and human-to-machine communication, but also ubiquitous machine-to-machine communication (see Fig. 10.1) occurs over the Internet. In the Internet of Things, objects of everyday life increasingly interact with each other autonomously (daCosta 2013).

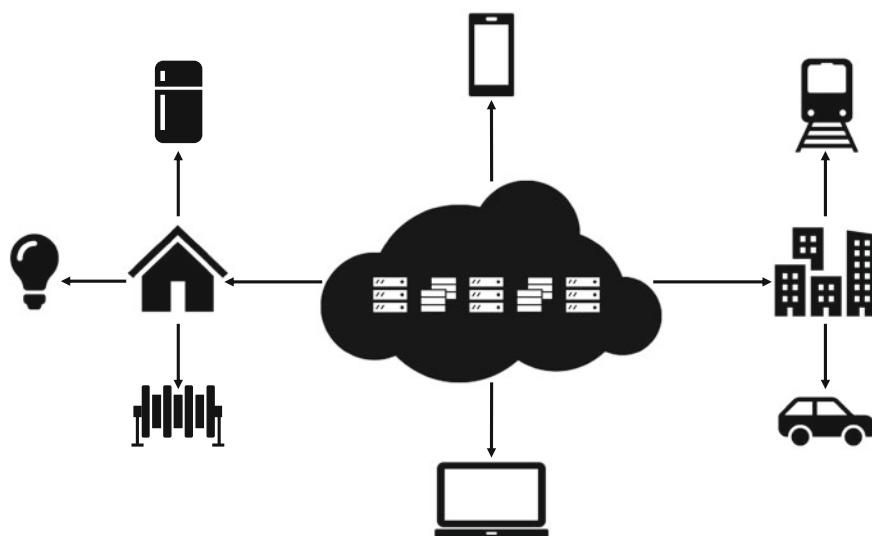


Fig. 10.1 Schematic of the Internet of Things

Although the Internet of Things is becoming one of the major trends that drives the development of technologies in the information and communications sector (Miorandi et al. 2012), there is no single, widely accepted definition of this concept. Several standardization bodies and organizations, such as the Institute of Electrical and Electronics Engineers (IEEE) (Minerva et al. 2015), the International Telecommunication Union (ITU) of the United Nations (International Telecommunication Union 2005), multinational software corporation SAP (SAP SE 2018), and the American multinational technological conglomerate Cisco (Cisco 2014), have each developed individual definitions of the Internet of Things. All of these definitions and the many more that exist, highlight the practitioners', governments', and researchers' diverse perspectives on this concept. Overall, most definitions of the Internet of Things can be regarded as either focusing on the everyday physical objects that are connected within a ubiquitous network (e.g., the definitions by the IEEE and SAP), the Internet-related aspects of the Internet of Things, such as Internet protocols and network technologies (e.g., the ITU definition), or the semantic challenges in the Internet of Things, such as storing, searching, and organizing large amounts of information (e.g., the definition by Cisco).

For the remainder of this book the Internet of Things will be referred to as a self-configuring, adaptive, complex network that interconnects "things" which have a physical and virtual representation over the Internet via standard communication protocols (Minerva et al. 2015). Drawing on this definition, the essential characteristics of the Internet of Things include:

- Interconnection of things: The Internet of Things interconnects "things" with each other, with "thing" referring to any physical object that is relevant to the Internet of Things' users.

- Connection of things to the Internet: The physical objects in the Internet of Things are connected to the Internet, which sets the Internet of Things apart from the physical objects organized in an intranet or extranet.
- Uniquely identifiable things: Each and every physical object connected to the Internet of Things possesses a virtual representation that makes it uniquely identifiable.
- Ubiquity: The Internet of Things network is available anytime and everywhere, when needed.
- Sensing (and actuation) capabilities: The physical objects connected to the Internet of Things have sensing capabilities, to a certain extent. Many physical objects also possess actuation capabilities, allowing the physical objects to interact with the immediate environment.
- Embedded intelligence: The physical objects in the Internet of Things possess a form of embedded intelligence (albeit often only rudimentary), which enables these physical objects to function as extensions of the human body and mind.
- Interoperable communication capabilities: The Internet of Things allows for communication based on standardized and interoperable protocols.
- Self-configurability: The Internet of Things is self-configurable, which means that it discovers new physical objects and services, as well as manages network and resource usage, independently.
- Programmability: The physical objects in the Internet of Things are programmable in that users can change and adapt their behavior without needing to change their physical representation.

Internet of Things

The Internet of Things is a self-configuring, adaptive, and complex network that interconnects "things" which have a physical and virtual representation on the Internet according to standard communication protocols. It entails nine essential characteristics: (1) interconnection of things; (2) connection of things to the Internet; (3) uniquely identifiable things; (4) ubiquity; (5) sensing (and actuation) capabilities; (6) embedded intelligence; (7) interoperable communication capabilities; (8) self-configurability; and (9) programmability.

10.1.2 A Brief History of the Internet of Things

To understand the Internet of Things concept, it is important to study its young, but eventful history. Despite being a relatively young concept that emerged in the late 1990s, the Internet of Things' roots can be traced back much further. A few researchers go so far as to argue that the Internet of Things' foundations were laid in the late 1920s to the mid-1930s when Nikola Tesla envisioned a "connected world" (Schoder 2018) and Sir Robert Alexander Watson-Watt discovered the radar (Minerva et al. 2015). While these pioneers' contributions to the foundations of our

modern and highly technological society cannot be underestimated, the Internet of Things' roots date back to the 1970s when the demand for large, mainframe computers started to shift to the smaller minicomputers and eventually to personal computers.

Following the strong miniaturization trend in the 1970s and 1980s, Mark Weiser, the then Chief Technologist at the Xerox Palo Alto Research Center (PARC), coined the phrase "ubiquitous computing" in the late 1980s. Ubiquitous computing describes a world in which a multitude of seemingly invisible computers surround people everywhere, at any given time (Weiser 1991). Weiser later summarized his idea of ubiquitous computing in his now famous 1991 article entitled "The Computer for the 21st Century" (Weiser 1991). Although ubiquitous computing certainly can be considered a predecessor technology of the Internet of Things, it is different in that ubiquitous computing still centers around the concept of computers, whereas the Internet of Things is not limited to computers but also comprises smart phones, wearable devices, and other physical objects.

The 1990s marked several important milestones for the Internet of Things. The year 1990 brought forth what many believe is the very first Internet of Things device. John Romkey and Simon Hackett connected a toaster to the Internet using Transmission Control Protocol/Internet Protocol (TCP/IP) in order to turn the toaster on and off (Romkey 2017). Furthermore, in 1999, Kevin Ashton, cofounder of the Auto-ID Center at the Massachusetts Institute of Technology, coined the term "Internet of Things" (Schoder 2018). At the time, Ashton's group was working on using radio frequency identification (RFID) and barcodes for supply chain management (Schoder 2018). Ashton envisioned a world in which computers would track and count everything. Moreover, the computers would also know everything about the things they track and count by autonomously collecting data about these things (Ashton 2009). Ashton later stated in an interview:

"I could be wrong, but I'm fairly sure the phrase 'Internet of Things' started life as the title of a presentation I made at Procter & Gamble (P&G) in 1999. Linking the new idea of RFID in P&G's supply chain to the then-red-hot topic of the Internet was more than just a good way to get executive attention." (Ashton 2009).

The 2000s – the new millennium – proved a crucial decade for the Internet of Things' evolution. In the year 2000, the South Korean electronics giant, LG, introduced the first internet-connected refrigerator, the LG Internet Digital DIOS. This refrigerator, which resulted from an internal LG research project that started in 1997, provided several informational functionalities (e.g., temperature) and management functionality (e.g., tracking foods inside the refrigerator). However, as a product, the refrigerator ultimately failed, since it was deemed unreasonably expensive and customers saw little value in the added functionality. During the early 2000s, several other devices that could connect to the Internet emerged (e.g., Ambient Orb by Ambient devices in 2002 and the Internet-connected mechanical rabbit, Nabaztag, by Violet in 2005), and the term "Internet of Things" started appearing in the mainstream media and books.

2005 marked the next important milestone for the Internet of Things when the United Nations' ITU published its first report on the topic (International Telecommunication Union 2005). This milestone was followed by the first European IoT Conference in 2008 and the U.S. National Intelligence Council citing the Internet of Things as one of six disruptive civil technologies that could potentially have an impact on the U.S. The hype around smart devices, such as smart phones and tablet computers, led to a rapid increase in internet-connected devices during the second half of the 2000s. As a result, Cisco's Internet Business Solutions Group estimated that the number of devices connected to the Internet has outgrown the number of living people at some point between 2008 and 2009 (Chen 2018). This moment is often referred to as the birth of the Internet of Things. Cisco also estimated that roughly 12.5 billion devices were connected to the Internet (compared to 6.8 billion living people) in 2010 and that this number would quadruple to 50 billion Internet-connected devices in 2020 (compared to an estimated 7.6 billion living people by that time).

In 2010, the Chinese government followed the European Union and U.S. in declaring the Internet of Things a key industry sector. Consequently, the then Premier of the People's Republic of China, Wen Jiabao, also announced the Chinese government's large investments in the Internet of Things and their goal to become a leader in this industry (Chen et al. 2018). In 2011, the information technology (IT) market research and analysis firm, Gartner, added the Internet of Things to its annual Hype Cycle (Gartner 2014), which indicated that the Internet of Things reached the so-called peak of inflated expectations in 2014 and would, by 2018, be expected to progress to the plateau of productivity within the next five to ten years. This signals more important steppingstones for the Internet of Things in the immediate future.

10.2 The Internet of Things: Technologies and Architectures

10.2.1 Enabling Technologies

Not one technology, but a combination of technologies jointly lay the foundation for the Internet of Things. Next to the Internet, which serves as the Internet of Things' backbone, a variety of other enabling technologies facilitate the Internet of Things. According to the ITU report on the Internet of Things cited earlier in this chapter, four kinds of enabling technologies seem most relevant to the Internet of Things: (1) tagging technologies, (2) sensor technologies, (3) smart technologies, and (4) miniaturization technologies.

Tagging Technologies

An essential feature of the Internet of Things is its ability to virtually track and count any physical object. Owing to the large number of potential objects that have to be

tracked, inexpensive and easy-to-use tagging technologies are very important for the Internet of Things. At first glance, a relatively uncomplex, cheap, and easy-to-use tagging technology that could be used for the Internet of Things are barcodes, which were developed in the U.S. in 1952 and which can be found on most retail products. Most barcodes used currently represent information by vertical lines of varying width and with varying spacing between each line. These vertical lines can, thus, convey only a very limited amount of static information, which cannot be easily changed, owing to barcodes being printed. Although different barcode designs can store different amounts of information, barcodes are frequently used to identify groups of products rather than each product individually. Take a box of milk from your nearby supermarket as an example. All the similarly branded boxes of milk from this supermarket will have the same barcode that convey exactly the same information. Furthermore, barcodes like other optical identification tags (e.g., Quick Response codes) require an optical reading device and a line of sight to convey their information.

RFID, which is closely connected to the notion of the Internet of Things, is a tagging technology aimed at addressing the limitations of traditional barcodes. A single RFID tag can store and transmit much more information than a barcode, up to two kilobytes of information for most tags. Consequently, RFID tags enable the tracking of many more objects. For instance, each box of milk from your local supermarket can be tracked individually. Barcodes constitute an optical tagging process, meaning that a person who wants to read a barcode will need a direct line of sight. RFID tags, in contrast, transmit their data via radio waves and therefore do not require a direct line of sight. Most RFID tags have a range of approximately one meter. In certain cases, this range could be limited to a few centimeters (e.g., for security reasons) or expanded to several meters. This RFID feature should not be underestimated, especially in industrial settings where several objects that are, for example, packaged together for transportation can be identified without having to tamper with the packaging. There are two main types of RFID tags, namely active and passive tags. The primary difference between them is their power source. Active RFID tags use an internal transmitter and power source, usually a battery; the internal power source powers the tag and allows it to transmit data without an external power source. Passive RFID tags do not use an internal power source, but draw energy from the reading devices that emit electromagnetic waves, which induce a current in the tag's antenna that produces just enough energy to send data to the reading device. Owing to their internal power source, active tags can usually transmit their data over a greater distance (more than 100 meters) than passive tags (up to 10 meters).

Although RFID tags have certain important advantages over barcodes and other optical identification tags, such as Quick Response codes, they have not yet succeeded in completely replacing optical identification tags, mainly owing to the cost per tag, which still far exceeds the costs of most optical identification tags. On average, an RFID tag costs around EUR 0.10, depending on the type of tag and the volume of tags ordered, compared to the cost of about EUR 0.01 or EUR 0.02 for a

single barcode, depending on the printing technique and the volume printed. However, a number of organizations have started to replace all their barcodes with RFID tags in order to track individual items. The Chinese convenience chain store, Bingo Box, for example, uses RFID tags in their staffless convenience stores, thereby enabling customers to check out their purchases without a cashier.

Sensor Technologies

Sensing, besides tagging, is a core capability of the Internet of Things that is driven by the rapidly decreasing costs for all kinds of sensors (Swan 2012). Sensors come in various sizes and form factors, can be wired or wireless, and exhibit a wide range of sensing capabilities (e.g., light, movement, pressure, temperature, smoke, proximity, acceleration). They can augment or complement human senses and collect data about the real world. Today, sensors are found almost everywhere around people in their digital societies, from modern mobile phones that contain at least proximity, acceleration, and Global Positioning System (GPS) sensors, to smart watches, which often contain sensors to measure a person's heart rate, to wearable sensor patches (i.e., "low-cost disposable patches that are worn continuously for days at a time and then discarded" (Swan 2012)). These sensing capabilities help provide various functionalities to homes, offices, factories, hospitals, and entire cities, which are increasingly enriched with sensors. However, a single, unconnected sensor has little value. For any sensor to be of value, the data collected by the sensor must be processed and interpreted. Within the Internet of Things, sensors are connected to the Internet, and these sensors' data are often sent to central cloud services where they are processed (see Chapter 7). Sensors become even more powerful and valuable when many of them are interlinked. For example, a wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices that use sensors to record physical quantities and, thus, map environmental conditions (Verdone et al. 2010). A WSN system consists of a gateway that allows wireless connection to wired systems and distributed nodes. WSN systems are a foundational technology for many Internet of Things applications, such as fully connected smart cities and the Industrial Internet of Things (see Section 10.3.3).

In many Internet of Things application scenarios, it does not suffice for physical objects to only sense information and report the data. Often, these physical objects are also expected to physically interact with other objects and people. Actuators provide this functionality. The actuators complement the sensors by giving physical objects the ability to interact with their immediate environments. The actuators range from automatic door locks to window shades, fire sprinklers, and many more. WSNs can be augmented with actuators, thereby forming a wireless sensor and actuator network (WSAN) (Verdone et al. 2010). The example in Fig. 10.2, for instance, comprises a temperature sensor that can detect unusual heat in case of a fire. This information is passed on to a control center, which processes and interprets the information, and eventually decides to turn on an actuator in the form of a sprinkler to extinguish the detected fire.

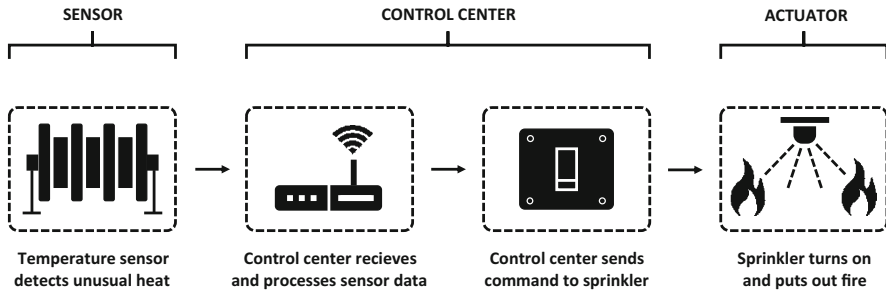


Fig. 10.2 Sensor to actuator flow (adapted from Misra (2017))

Smart Technologies

Smart technologies are a third category of enabling technologies for the Internet of Things. As stated, increasingly more sensors are deployed in people's various immediate environments, which produce a rapidly increasing amount of data that must be processed and interpreted in order to provide services and applications via the Internet of Things. Within the currently existing Internet of Things paradigm, most of the data produced by the Internet of Things is sent to cloud services, which process the data. However, even for cloud services with their apparently unlimited computing resources and for the underlying networks, the amount of data produced by the hundreds of billions connected physical objects and their sensors will be overwhelming (Bassi et al. 2013). The only solution for this dilemma is to move processing from the heart of the Internet (i.e., cloud servers) to the edges of the network (i.e., smart things). Physical objects with embedded intelligence (i.e., a certain amount of processing power and the ability to respond to external stimuli) will relieve cloud servers and the network from processing a large portion of the data produced by the Internet of Things and enable these smart "things" to decide and act independently (International Telecommunication Union 2005). The concept of moving the processing power from the Internet's heart to the network's edges is also referred to as edge computing (see Chapter 8).

Miniaturization Technologies

In order to unobtrusively embed intelligence into almost all kinds of everyday physical objects, the computer chips and sensors must become increasingly smaller (International Telecommunication Union 2005). Owing to astounding advances in miniaturization and nanotechnology during the last decades, technology has progressed, for instance, from bulky vacuum tubes that consume a lot of energy and produce a lot of heat to process information, to smaller and more efficient transistors, and to tiny integrated circuits that operate on a nanometer scale (International Telecommunication Union 2005). Consequently, computer chips are nowadays found in almost every single electronic device. Without such miniaturization progress, the Internet of Things in its current form and anticipated future form would not be possible.

Table 10.1 below summarizes the enabling technologies for the Internet of Things, as well as their purposes, examples, and standard challenges.

Table 10.1 Summary of enabling technologies for the Internet of Things

Enabling Technologies	Purposes	Examples	Standard Challenges
Tagging technologies	Identify and track individual physical objects	Active/passive RFID tags	Cost of tags are still considerably higher than printed barcodes
Sensor technologies	Collect data about the real world and augment human senses	Temperature sensor, proximity sensor, GPS	Processing and making sense of the vast amounts of collected sensor data
Smart technologies	Provide processing capabilities to physical objects	Microprocessors	With improved capabilities, smart technologies require an increasing amount of energy
Miniaturization technologies	Shrink information technology such that it can fit into everyday objects	Seven nanometer transistor manufacturing processes	As miniaturization continues, it becomes increasingly difficult to produce increasingly smaller things

10.2.2 Core Concepts

Internet-connected physical objects with embedded intelligence, also known as smart things, are an integral aspect of the Internet of Things. To this end, two types of things, namely smart devices and smart objects, can be differentiated.

Smart Devices

A smart device is a portable electronic device that enables its users to access a variety of local and remote services. Standard examples of smart devices include mobile phones, tablet computers, smart watches, and Internet-connected TVs. Although the term "smart device" only gained widespread recognition with the triumphal success of smart phones in the late 2000s, the idea of smart devices has already been around for decades. For example, in 1991, when Mark Weiser published his vision of ubiquitous computing, he noticed a trend for how humans interact with computers. This trend started in the mainframe era during which many people used one computer and expanded to one computer for each person in the personal computer era (i.e., the 1990s and 2000s). Then, this trend expanded further to the current reality of many computers for each person. Consequently, Weiser described three electronic devices that would allow people ubiquitous access to computing resources: tabs (i.e., wearable, centimeter-sized electronic devices, such as smart watches and fitness trackers), pads (i.e., handheld, decimeter-sized electronic devices, such as smart phones and tablets), and boards (i.e., meter-sized displays, such as vertical and wall-mounted computers). In a certain sense, these tabs, pads,

and boards envisioned by Weiser can be considered the precursors of modern smart devices. A distinct feature of smart devices is that they usually belong to and are used by a specific person. Think of a smart phone, for example: In most cases, one person owns and uses it. Smart devices are usually multi-functional and mobile (although in varying degrees), can dynamically discover new or extant services, and have intermittent access to resources. Most smart devices today also exhibit location-awareness and self-awareness to a certain extent. The term "smart device" is usually associated with planar devices that are at least watch-sized and provide, to a certain extent, a display (Poslad 2011). This is also true for the tabs, pads, and boards that Weiser described. However, the smart device concept also comprises other, more diverse types of devices, such as smart dusts or smart skins. Smart dusts, for example, are a collection of miniaturized smart devices with sensing and processing capabilities, but without an integrated display (Warneke et al. 2001). These smart dusts are often only fingertip-sized and can be deployed to collect data unobtrusively within an area (e.g., to monitor engines, the environment, or wildlife) (Warneke et al. 2001). Smart skins, on the other hand, are fabric-based, non-planar, flexible, and stretchable smart devices (Benight et al. 2013) that try to imitate human or animal skin and hence their predominant application area is in health care monitoring and prosthetics (Benight et al. 2013).

Smart Device

Smart devices are portable multi-purpose information and communication technology (ICT) devices that enable access to several application services located on the device or remotely on servers. They are usually owned and used by one person. Examples of smart devices include laptops, mobile phones, and tablets.

Smart Objects

Smart objects, often synonymously referred to as smart things, are physical objects that can autonomously interact with humans, as well as other objects and smart objects in their environment. Smart objects can recognize and collect data about their environment via implemented sensors, they can process and store this data owing to embedded microprocessors and memory, and they can also communicate via network and user interfaces (see Section 10.2.1) (Kortuem et al. 2009). A few examples of smart objects are smart speakers like Amazon Echo, Apple HomePod, and Google Home, smart refrigerators that can track the products they store, smart traffic signs that can automatically adjust to environmental conditions, and smart shipping containers that track, secure, and monitor themselves.

Within organizational contexts, one can differentiate between activity-aware, policy-aware, and process-aware smart objects (Kortuem et al. 2009). Activity-aware smart objects have the ability to collect and record information relating to

work activities and how they are used (e.g., on a construction site, tools can record when, by whom, and for how long they are used in order to establish a pay-per-use pricing model). Policy-aware smart objects are activity-aware smart objects that understand and process events and activities regarding organizational policies (e.g., a smart truck that measures and records driving hours and understands legal limitations, such as a truck driver who may only drive a truck during permitted hours), whereas process-aware smart objects can understand and support organizational processes, as well as the individuals involved in these processes (e.g., a set of tools in a factory that understands the assembly process and signals when to use which tool for which purpose within the process).

Smart Object

Smart objects are physical objects with an ability to recognize, process, and exchange data, act autonomously, adapt flexibly to specific situations, and interact with humans, as well as other objects and smart objects. Standard examples of smart objects include intelligent light bulbs, mugs, and heaters.

The literature on the Internet of Things often does not explicitly differentiate between smart devices and smart objects, and often uses both terms interchangeable or only refers to smart objects. Consequently, the partial overlap between the two terms "smart device" and "smart object" dealt with in this in this chapter must be noted (see Fig. 10.3). Smart watches and other wearable devices, which are often referred to as smart objects, but also fit the above-mentioned definition of smart devices, exemplify this overlap. Nevertheless, smart objects represent the broader class of regular objects, whereas smart devices represent the narrower class of electronic devices. Moreover, and in contrast to smart devices, smart objects do not necessarily have to be portable or have a specific owner.

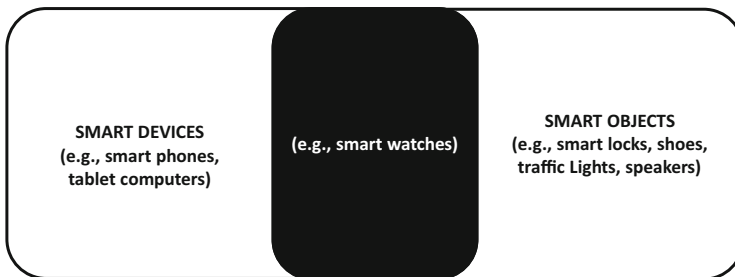


Fig. 10.3 Conceptual overlap of smart devices and smart objects

Smart Environments

Many smart objects distributed in a physical world can together form smart environments. Smart environments are physical worlds that are interwoven with interconnected smart objects and therefore have the ability to monitor and interact with their surroundings (López et al. 2011). An essential feature of smart environments is that they are context-aware: They can sense and interpret contexts in order to provide a variety of context-dependent information and services. Smart environments are what most people associate with the term "Internet of Things," although the Internet of Things is a more holistic concept comprising many more things than only smart objects (e.g., smart devices) and many different types of smart environments. Smart homes, smart cities, and smart factories, to name but a few, are classic application areas of smart environments.

Although the smart environment concept is closely related to the ubiquitous computing concept and although both terms are often used interchangeably, the two concepts must not be confused with each other. In contrast to ubiquitous computing, which refers to a physical world that is interwoven with computing devices, smart environments refer to a physical world interwoven with smart objects. In a sense, smart environments thus represent the next step and a more radical form of ubiquitous computing.

Smart Environment

A smart environment is a physical world interwoven with a multitude of sensors, actuators, displays, and computing elements, seamlessly interacting with everyday objects of people's lives and connected via a continuous network.

10.2.3 Architecture Models

Similar to the Internet of Things' diverse definitions, there is no single widely accepted architectural model for the Internet of Things. Instead, a variety of standardization bodies and organizations, such as the IEEE, Cisco, and the Internet of Things World Forum, have each put forward a specific architectural model, which represent their individual views of the Internet of Things and exhibit different foci. This Section therefore presents a selection of prominent architectural models for the Internet of Things, which should by no means be considered complete or exhaustive.

The two most basic forms of architectural models for the Internet of Things are the three-layer and five-layer models, which, as their names imply, comprise either three or five layers that form the Internet of Things (see Fig. 10.4). The three-layer Internet of Things model consists of a perception layer at the bottom, an application layer at the top, and a network layer in between the perception and application layers

(Sethi and Sarangi 2017). The perception layer, which is occasionally also referred to as the recognition or sensing layer, is responsible for sensing and collecting useful information about the environment. All the smart objects are situated in the perception layer. The network or communications layer, which forms the Internet of Things' backbone, is responsible for transmitting the information collected at the perception layer securely to the application layer and also for transferring the information produced at the application layer back to the perception layer. All the network technologies, such as WSNs, WSANs, and the Internet infrastructure itself, are located in the network layer. The last and topmost layer in the three-layer Internet of Things architecture model is the application layer. This layer's main task is providing services to the users of the Internet of Things. The application layer is usually the most prevalent among users, as they often interact directly with the applications in this layer.

The five-layer Internet of Things architecture model builds directly on the three-layer model and was developed to account for the anticipated future developments of the Internet of Things (see Fig. 10.4) (Muntjir et al. 2017). While the three-layer model encapsulates the Internet of Things' core idea (Sethi and Sarangi 2017), the five-layer model extends this core idea by introducing two new layers and refining a few of the layers that are already included in the three-layer model. The five-layer model comprises a perception layer at the very bottom, followed by a transport layer and a processing layer. The two topmost layers in the five-layer model are the application layer and the business layer at the very top. The five-layer model's perception layer corresponds directly to the three-layer model's perception layer and fulfills the same function. Analogously, the transport layer on top of the perception layer corresponds to the network layer in the three-layer model. The transport layer's main task is transferring data from the perception layer to the processing layer and vice versa. The processing layer, which is occasionally also referred to as the middleware layer, is a new layer that is not part of the three-layer model. The processing layer stores and processes the vast amounts of data that are produced by the Internet of Things and are received from the perception layer via the transport layer. Technologies, such as databases and cloud computing, can be situated in this layer. Similar to the three-layer model, the application layer in the five-layer model defines and delivers the application services to the Internet of Things' users. Finally, the topmost layer, the business layer, manages the entire system that comprises the Internet of Things. This task includes managing applications, business models, and user privacy. The three-layer and five-layer models' application layers, although quite similar, are not identical, as the application layer in the three-layer model comprises more responsibilities than the application layer in the five-layer model. A few of these responsibilities (e.g., data processing) can be found in the processing and business layers. Simultaneously, the five-layer model's processing, application, and business layers also include more than the application layer in the three-layer model (e.g., user privacy and business models).

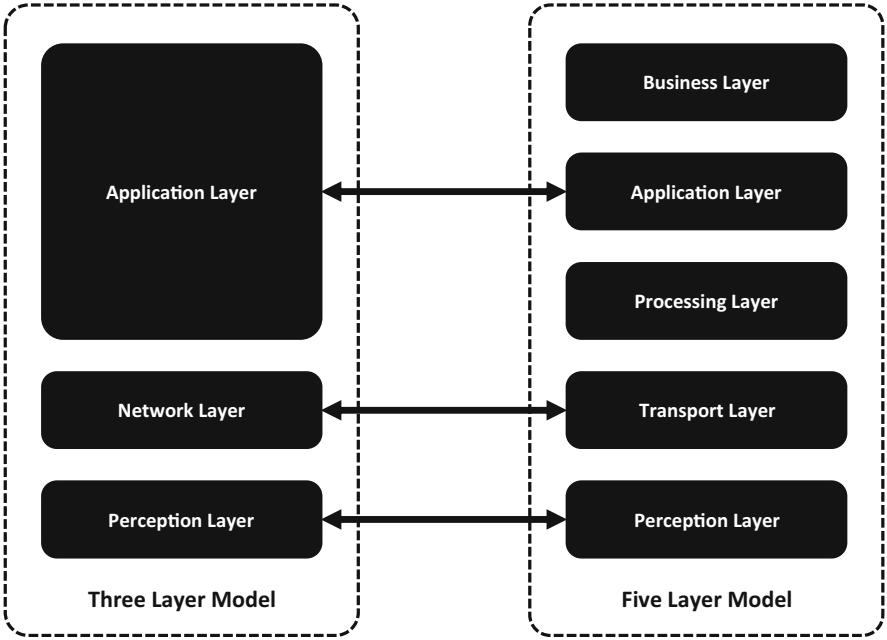


Fig. 10.4 Comparison of the three-layer and five-layer Internet of Things architectural models

The Internet of Things World Forum is an annual industry event hosted by Cisco. Following discussions at the forum, the IoT World Forum Architecture Committee was formed and tasked to develop a reference architecture for the Internet of Things (Cisco 2014). The committee mainly consists of large, international telecommunications enterprises, such as Cisco, IBM, Intel, Oracle, PARC, Samsung, SAP, and Qualcomm, and also includes well-known enterprises from other industries, such as Gartner, General Electric, General Motors, and Starbucks. The IoT World Forum Architecture (see Fig. 10.5) consists of seven layers or levels in total. Similar to the three-layer and five-layer models, the bottommost level in this model is the physical devices and controllers level, which is where all the physical (smart) objects that participate in the Internet of Things are located. The physical devices and controllers level is followed by the connectivity level, which is responsible for the reliable and timely transmission of data and, thus, corresponds to the previously discussed models' network and transport layers. On top of the connectivity level sits the edge computing level. This level's main task is to convert a flood of data into information that can be stored and used for higher level processing at the data accumulation level which is located directly above the edge computing level. From level 1 (physical devices and controllers) to level 3 (edge computing) the collected data is in a constant state of flow. The data accumulation level puts this data at rest so that

applications can use it when needed on a non-real time basis. Level 5, the data abstraction level, provides data abstraction functions to the upper levels, which enable upper levels to handle the flood of data produced by the Internet of Things. The sixth level in this architecture is the application level, which is where information interpretation occurs via the diverse applications that reside at this level. This application level corresponds to the application layers in the three-layer and five-layer models. The collaboration and processes level is this architecture's last, top-most level. This level goes beyond the pure technical aspects of the Internet of Things and represents its socio-technical aspects. The collaboration and processes level includes people and processes that connect the multiple applications provided by the Internet of Things.

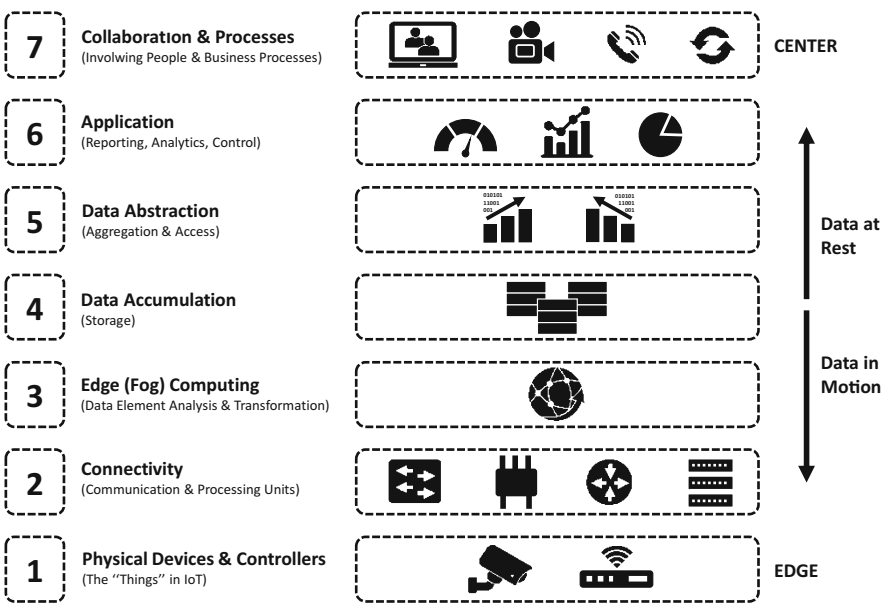


Fig. 10.5 Overview of the Cisco Internet of Things Reference Model (adapted from Cisco (2014))

Another notable reference model for the Internet of Things is the IoT-A (short for Internet of Things Architecture) model that was developed by a consortium of industry and research partners within the European Union's Framework Programme for Research and Technological Development (Carrez et al. 2013). The IoT-A model is not based on a layered structure and much more complex than the three previously presented models. Instead, the IoT-A model consists of five submodels that interact with each other (see Fig. 10.6).

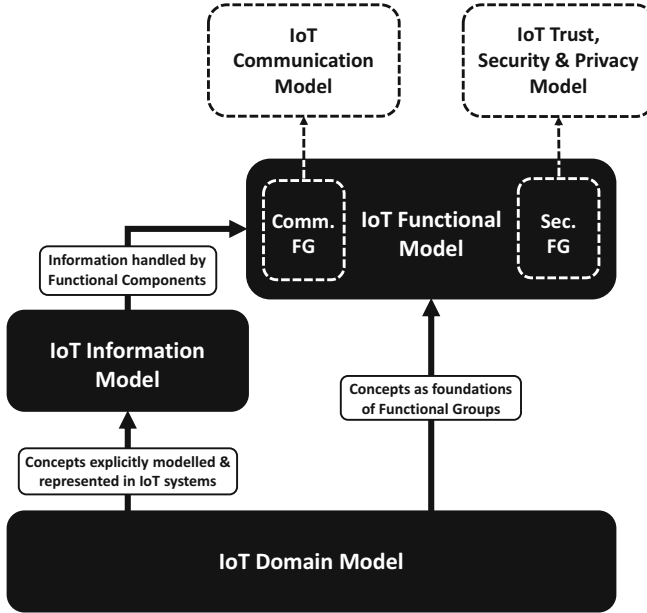


Fig. 10.6 Overview of the IoT-A reference model (adapted from Carrez et al. (2013))

Owing to each submodel's extent and complexity, an in-depth description of the IoT-A model is beyond the scope of this chapter. Therefore, each submodel will be briefly introduced and interested readers will be directed to this chapter's further reading section, which also includes readings on the IoT-A model. The IoT-A model's first and foundational submodel is the IoT domain model, which is responsible for introducing the Internet of Things' basic concepts (e.g., devices, Internet of Things services, and virtual entities) and their relationships. The next submodel is the IoT information model, which describes the structure (attributes, relations, etc.) of relevant information within the context of the Internet of Things. The IoT functional model is the third submodel in the IoT-A reference model. The functional model follows the functional decomposition idea in order to break down the complexity of systems that are built on the IoT-A reference model. The functional model defines several functional groups, including, for example, device, communication, management, security, application, service organization, and IoT process management. The IoT communication, as well as the IoT trust, security, and privacy submodels are the last two submodels described here and are themselves submodels of the IoT functional model. The IoT communication model's main task is defining the communication paradigms that support communication between the basic concepts defined by the IoT domain model. The IoT trust, security, and privacy model

identifies the desired trust, security, and privacy properties and defines several guidelines to ensure these properties.

10.3 Internet of Things Applications

The Internet of Things has potential to radically transform many areas of people's lives. Important application areas of the Internet of Things are, among others, smart homes, smart cities, the Industrial Internet of Things, the healthcare sector, and the energy sector. Although this is certainly not a complete list of important Internet of Things application areas, the key technologies, as well as exemplary scenarios for each of the five aforementioned application areas, will be described in this section of the chapter.

10.3.1 *Smart Homes*

Nowadays, most homes in the developed and developing countries are connected to the Internet (Statista 2019c). Many modern households currently have a variety of Internet-connected smart devices and objects, such as multiple smart phones, smart TVs, remote-controlled light bulbs, and many more. Consequently, smart homes have become one of the most popular Internet of Things application areas. Smart homes are homes that are interspersed with smart objects and smart devices, providing these homes with an awareness of what occurs inside them and the ability to act either upon their residents' commands or autonomously in case of specific events. Based on these characteristics, smart homes can be considered a type of smart environment. Smart homes mainly impact three areas of living: comfort, resource usage, and security (Bassi et al. 2013). Smart homes strive to improve living comfort, while simultaneously decreasing resource usage and increasing security (e.g., via Internet-connected surveillance cameras and remote-controlled lights).

Smart Homes

Smart homes are a kind of smart environment. They are homes that are interspersed with smart objects and devices, which allow for homes to be aware of what happens inside. Smart homes strive to improve living comfort, while they simultaneously decrease resource usage and increase security.

The market offers many smart home products that aim to improve comfort and quality of living. One of the first products that falls into this category is smart lights, which can be operated with a remote control (e.g., via a smart phone or tablet application) in order to turn them on or off, and change their brightness or color. Smart lights can also be programmed, for example, to automatically turn on or off on certain days at specific times. Popular smart light systems include Philips Hue, TRÅDFRI by IKEA, and LIGHTIFY by OSRAM, to name but a few. Another category of smart home products that aims at improving comfort is smart speakers, which offer a variety of services, such as playing music, receiving information from the web, and accessing other Internet-based services via voice control. Smart speakers have recently become the top trend in the industry, following technological advances in artificial intelligence and voice recognition, as well as the popularity of Amazon's smart speakers, Echo and Echo Dot, which are driven by Amazon's Alexa artificial intelligence. Smart speakers regularly act as smart home control centers and allow for controlling other smart objects and devices via voice commands. Other well-known smart speakers include Google Home, which allows access to Google services and the Google Assistant artificial intelligence, and Apple HomePod, which is driven by Apple's artificial intelligence, Siri. Although various manufacturers offer many more smart speakers, they mostly connect to one or multiple artificial intelligences of the big consumer technology providers (i.e., Amazon, Apple, Google, and Microsoft). While smart lights and smart speakers are currently the main kinds of smart objects for improving comfort in a smart home, a multitude of other smart objects, such as intelligent refrigerators, autonomous vacuum cleaners, smart kitchenware, and smart air humidifiers are available to consumers.

Another important aspect of smart homes is resource optimization by deploying smart objects in people's homes. A smart thermostat is an example of a smart object that is used for resource optimization. Smart thermostats allow their users to adjust the heating and air-conditioning in their homes via remote control. Besides more intuitive features, such as comfortably setting the preferred temperature, smart thermostats like Google Nest also have the ability to learn about the home residents' behaviors and temperature preferences, and adjust heating (or cooling) accordingly. These features enable smart thermostats to optimize heating and cooling, thus reducing unnecessary and wasteful resource usage. Smart thermostats can also provide comprehensive reports to their users, offering further potential to reduce resource usage by, for example, increasing the residents' awareness of the amount of resources used.

Another example of smart objects that aim at reducing resource usage are smart meters. In contrast to smart thermostats, smart meters monitor, record, and report the electricity consumption to a home owner's electricity supplier. On the one hand, smart meters provide a means to adequately measure actual power consumption in real time, thus eliminating the need for estimated monthly electricity bills and costly subsequent payments. On the other hand, and similar to smart thermostats, smart meters allow residents to develop an electricity usage awareness, thus creating

incentives to save electricity. Recent studies show that smart meters can reduce energy consumption by 3-5% each year (McKerracher and Torriti 2013).

Lastly, smart homes also aim at increasing security by deploying complex security systems in order to prevent theft and burglary. Smart door and window locks, for example, can be controlled remotely from a smart phone and other devices or provide individual and temporary access to certain people. The state of most smart locks can also be checked and monitored from afar, giving users real-time information about whether a door or window had been unlocked or opened. A few of the more complex smart lock systems further enhance this basic functionality by, for example, providing real-time video footage in case a lock is tampered with, unlocked, or simply if a movement outside a home is detected. However, security enhancements in smart homes do not only cover theft and burglary; these enhancements also include the safety of residents by, for example, deploying smart smoke detectors and sprinklers. Picking up on the previous example, a combination of smart thermostats and smart smoke detectors could detect a fire, set off a smart sprinkler to extinguish the fire, and notify the local fire department, all without or with minimal human interaction.

In a fully realized smart home, all the different smart objects and devices work together in a well-orchestrated manner in order to provide higher comfort, reduce resource usage, and provide increased security. Consequently, standards and protocols, occasionally also called home automation standards or protocols, that allow for the seamless integration and interaction of the diverse smart objects and devices from manifold manufacturers that can be found in a smart home, are required. To this end, a variety of smart home standards and protocols have been developed over the past years. Out of the currently existing home automation protocols, Z-Wave and Zigbee are the most popular.

In 2001, the Danish firm, Zensys, developed the Z-Wave protocol, which was later acquired by the U.S.-based company, Sigma Designs, in 2008. Z-Wave uses a low-energy radio frequency-based mesh network in the 800 to 900 MHz range to directly connect individual network participants (e.g., smart objects and devices). It allows data transmission rates of up to 100 kilobits per second, with a range of about 24 meters indoors, up to 100 meters outdoors, and a maximum of 232 devices per Z-Wave network. Owing to each Z-Wave network being assigned a unique 32-bit ID, a maximum of 4.2 billion Z-Wave networks is permissible.

In comparison, Zigbee is a standard of communication protocols that is based on the IEEE 802.15.4 specification, which the Zigbee Alliance, an industry consortium, developed in 2004. Similar to Z-Wave, Zigbee provides a low-power radio-frequency-based mesh network with comparable range. However, unlike Z-Wave, Zigbee networks usually operate on the 2.4 GHz band, which allows for data transmission rates of up to 250 kilobits per second. Technically, a Zigbee network supports up to roughly 65,000 devices in each network. Table 10.2 provides a more detailed comparison of Z-Wave and Zigbee.

Table 10.2 Comparison of pertinent smart home protocols

	Z-Wave	Zigbee
Developer	Zensys/Z-Wave Alliance	Zigbee Alliance
Year developed	2001	2004
Network type	Mesh	Mesh
Frequencies	800-900 MHz	2.4 GHz
Transmission rates	100 kB per second	250 kB per second
Range	24 meters (indoors) 100 meters (outdoors)	10-100 meters
Maximum number of devices per network	232	65,000

10.3.2 Smart Cities

Smart cities are another important application area for the Internet of Things. The smart city market is expected to reach a size of \$400 billion by 2020 (Bassi et al. 2013; Ove Arup & Partners 2013). Although there is no single, clear, and widely accepted definition of a smart city, the general consensus seems to be that a smart city is a sustainably developed urban area that offers a high quality of living (Bassi et al. 2013). Transforming a regular city into a smart city requires changes across all city spheres, including its economy (e.g., innovative spirit, a flexible labor market, and the ability to transform), people (e.g., high level of qualification, creativity, and open-mindedness), governance (e.g., public and social services, and transparent governance), mobility (e.g., sustainable, innovative and safe transport systems, and availability of an ICT infrastructure), environment (e.g., attractiveness of natural conditions and sustainable resource management), and living (e.g., cultural facilities, individual safety, housing quality; see Fig. 10.7) (Giffinger 2007). Obviously, not all of the smart city's aspects can be achieved with or supported by the Internet of Things. However, to outperform non-smart cities in these six key areas, smart cities must not only possess strong human and social capital, but also a modern and pervasive ICT infrastructure (Bassi et al. 2013).

Smart Cities

Smart cities are sustainably developed urban areas that offer a high quality of living. They outperform non-smart cities in the spheres of economy, people, governance, mobility, environment, and living through a combination of strong human and social capital, and a modern, pervasive ICT infrastructure.

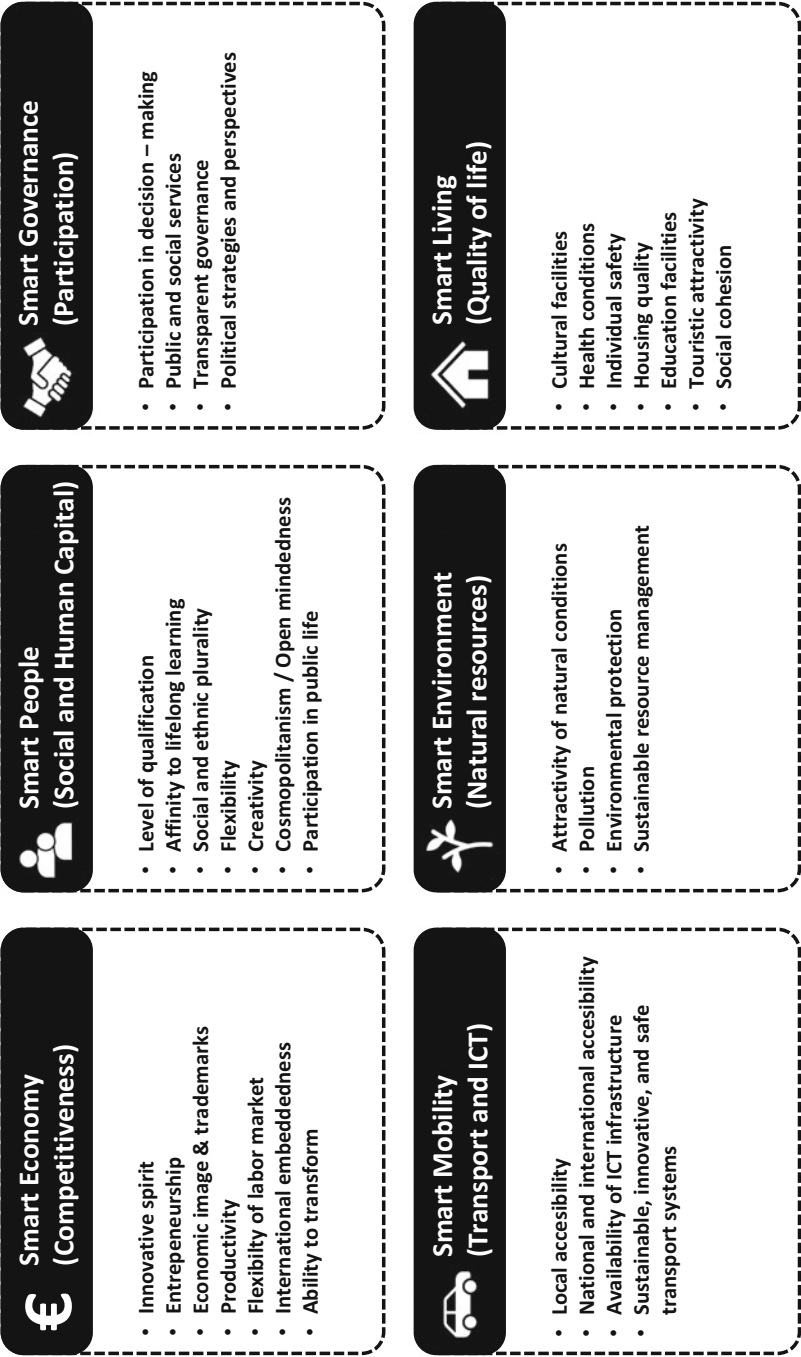


Fig. 10.7 Characteristics and features of a smart city (adapted from Giffinger (2007))

Since a detailed discussion of all six key areas is beyond the scope of this chapter and since certain key areas will be discussed later in this section, the next paragraphs provide a few examples of smart cities and projects aimed at transforming cities into smart cities. Where necessary, the key areas will be referenced.

In 2015, the U.S. Department of Transportation launched its Smart City Challenge in search of ideas for highly innovative and integrated transportation systems for mid-sized cities in the U.S. The goal was to “use data, applications, and technology to help people and goods move more quickly, cheaply, and efficiently” (Department of Transportation 2016). The initiative received 78 submissions from mid-sized U.S. cities, of which the following seven were selected for a second phase: Austin (Texas), Columbus (Ohio), Denver (Colorado), Kansas City (Missouri), Pittsburgh (Pennsylvania), Portland (Oregon), and San Francisco (California). Although the Smart City Challenge itself focused on improving transportation and, thus, mainly addresses the mobility area, many of the finalist cities developed concepts that go beyond the mobility area only. For example, a few of the five strategic goals Columbus defined in its application also address other key areas, such as governance and environment: Better connecting citizens to human services, encouraging greater use of sustainable transportation, improving access to jobs, reducing freight truck congestion using smart logistics, and providing real-time traffic information to improve commuter mobility. Consequently, Columbus plans to develop a single mobile application that enables residents, as well as visitors, to pay for all kinds of public transportation services in the city, including not only metros and busses, but also parking, car-sharing, ride-hailing, and even bike-sharing. As Columbus is a major logistics hub, truck congestion poses a serious problem to the city. Thus, Columbus also plans to develop a system that allows two or more semi-autonomously driven trucks to closely follow each other in order to ease congestion, save fuel, and increase safety.

Songdo in South Korea is another interesting smart city project that, although addressing many of the six key areas, exhibits a strong focus on the environment. Songdo is a 100,000 person, built-from-scratch smart city located southwest of Seoul, South Korea’s capital, that was completed in 2015. Although there are already technology-packed apartments throughout South Korea, the vast majority of apartments in Songdo further qualify as smart homes. Residents can, for example, use built-in communication systems to video call their neighbors or participate in remote classes. Hundreds of sensors have been deployed throughout the city, whereby nearly all the city’s aspects are monitorable and measurable, from traffic flow to energy use to crime. Residents can, for instance, also be notified by the central home communication systems when it is time to leave in order to catch the next bus. Moreover, the city was designed such that most residents can walk to their work place vs. having to use a car. Moreover, Songdo has no waste disposal trucks, as household waste is automatically transported from each apartment to a central waste disposal facility via an automated underground waste transport system. Overall, more than 100 buildings and 2 million square meters of space in Songdo are

LEED¹ certified, making it one of the most environmentally friendly cities in the world.

In Germany, the IT industry association, Bitkom, and the German *Städte- und Gemeindeverbund* initiated a digital city competition in 2016 (Strehmann 2017). The participating cities' concepts were evaluated according to their impact on education, data platforms, energy and environment, society, health, trade, ICT infrastructure, security, transport, and governance. Within the competition, the five German cities Darmstadt, Heidelberg, Kaiserslautern, Paderborn, and Wolfsburg scored the highest, with the city of Darmstadt ultimately winning the competition. Darmstadt's smart city concept aims at turning the city into a digital model city that serves as a role model for other cities in Germany. The concept defines the five core objectives: (1) value for citizens, (2) participation by citizens, (3) future orientation, (4) sustainability, and (5) security, as well as three main areas of action: (1) mobility and environment, (2) digital services and society, and (3) economy and technology (Roland Berger 2018). All smart city projects carried out or supported by the city of Darmstadt must address these core objectives and fall under at least one area of action. Current projects within Darmstadt's strategy include extensive investments in the local ICT infrastructure, designated programs in schools to prepare children for a digital society, online municipal services for residents, such as a chatbot for obtaining information and performing simple municipal tasks online, an online platform that allows residents to participate in the political decision-making, a smart waste system, which notifies waste disposal services when waste must be collected, smart lights, which adapt street lightning to the daily light conditions, as well as smart traffic systems that are able to monitor and optimize traffic flows. Other German cities with notable smart city ambitions include Berlin with its Smart City Berlin initiative and Hamburg with its smartPORT project.

10.3.3 The Industrial Internet of Things

The Industrial Internet of Things is yet another important application area of the Internet of Things and an extension of the Internet of Things concept. The Industrial Internet of Things refers to the application of Internet of Things technologies, such as certain smart objects and devices within an industrial setting, in order to promote goals that are unique to industry (Boyes et al. 2018). Occasionally, people also refer to it as the Industrial Internet or Industry 4.0 to underpin its importance as a fourth industrial revolution (the first is the production mechanization by means of steam and water power in the late 18th century, the second is the introduction of electricity, assembly lines, and mass production at the beginning of the 20th century, and the third is the production computerization and automation that commenced in the

¹LEED (short for Leadership in Energy and Environmental Design) is a worldwide certification program for environmentally friendly buildings.

1950s). Today, the Industrial Internet of Things already has a significant impact on the global economy (Daugherty et al. 2015), covering industries, such as energy, manufacturing, agriculture, health care, retail, logistics, and aviation that produce about 62 percent of the G20 nations' gross domestic product (Daugherty et al. 2015). Nonetheless, the Internet of Things' impact on the global economy will expectedly continue to increase and the Industrial Internet of Things will itself create a global market worth at least \$500 billion by 2020 (Daugherty et al. 2015).

Industrial Internet of Things

The Industrial Internet of Things is the application of Internet of Things technologies (e.g., smart objects and smart devices) within an industrial setting in order to promote goals that are unique to industry. It covers diverse industries, such as energy, manufacturing, agriculture, health care, retail, logistics, and aviation.

A key promise of the Industrial Internet of Things relates to improving operational efficiency (i.e., improving asset utilization, increasing productivity, and cutting down on operational expenses). Consequently, manufacturing firms were among the first to adopt the Industrial Internet of Things to expedite automation and flexibility (Daugherty et al. 2015). A smart factory is interspersed with sensors and smart objects that are connected to each other in a factory-spanning network (Lucke et al. 2008). These sensors and smart objects allow for the sensing and tracking of goods (e.g., via RFID tags), production flows, and providing additional context information that, combined, provide a smart factory with the ability to develop context awareness. Drawing on this context awareness, smart factories are capable of supporting humans and machines in the production process in order to increase efficiency or even enable machines to produce certain goods completely autonomously. Furthermore, the gathered information can be used to analyze and optimize production flows, and manage the inventory. Sensors embedded in machinery and Big Data analytics can also be leveraged to anticipate defects and enable an organization to engage in predictive maintenance to avoid unnecessary downtimes or shutdowns of entire production plants (Daugherty et al. 2015).

Smart Factory

A smart factory is a factory interspersed with sensors and smart objects that are connected to each other in a factory-spanning network. This network allows for sensing and tracking goods, as well as production flows, and provides additional context information. Smart factories have several advantages over traditional factories and, for example, allow for optimizing production flows and inventory management.

The British firm, Thames Water, exemplifies the Industrial Internet of Things' capabilities. Thames Water is the largest drinking water provider and wastewater recycler in the United Kingdom and uses a combination of sensors, remote communication, and Big Data analytics to engage in predictive maintenance (Gilchrist 2016). Consequently, Thames Water is able to anticipate equipment failure and respond quickly to critical situations that could not only cause damage to its business, but also impair the supply of fresh drinking water to the people in the United Kingdom. Similarly, Apache Corporation, a U.S. petroleum and natural gas exploration and production firm, uses predictive maintenance to predict onshore and offshore oil pump failure (Daugherty et al. 2015). Apache Corporation estimates that the industry-wide improvement of oil pump performance by as little as 1% could increase oil production by half a million barrels per day, resulting in additional revenue of \$19 billion a year. However, the potential of the Industrial Internet of Things reaches far beyond improvements in operational efficiency. For example, the Industrial Internet of Things also provides a foundation for new products and services and, therefore, the realization of entirely new business opportunities.

Researchers expect that the Industrial Internet of Things will have four evolutionary stages (see Fig. 10.8), ranging from industry's near-term adoption in order to realize near-term benefits, such as increased operational efficiency (stage 1) and, as suggested above, opportunities for new products and services (stage 2), to long-term structural changes in the economy, which will first lead to an outcome economy (stage 3), followed by an entirely autonomous pull economy (stage 4).

In an outcome economy, companies no longer focus on selling products and services to their customers based on the products' and services' face value. Instead, the company's business focus shifts toward delivering measurable results, constituting real benefits, to their customers (World Economic Forum 2015). The agricultural sector is a classic example of such a shift. Instead of selling agricultural machinery at face value, the machinery manufacturers could sell their products based on the yield per acre that farmers achieved with their help (Klewes et al. 2017). However, realizing an outcome economy is a challenge that requires firms to have a much better understanding of their customers' needs and how these needs change within different contexts. Furthermore, firms must be able to quantify the delivered results in real time. The better understanding and the real-time results quantification can only be achieved by means of the vast amount of data collected by an unprecedented number of smart objects and devices, as well as Big Data analytics that are brought forth by the Internet of Things. However, making sense of the amount of data necessary for realizing an outcome economy still poses a serious challenge to current technology and it will require several years before a stage is reached where an outcome economy becomes possible and feasible.

Even further ahead in the future, the Industrial Internet of Things' fourth stage will lead to a fully autonomous pull economy (World Economic Forum 2015). In such an economy, Internet of Things technologies will be deeply implanted in every industry, which enables firms to recognize the demand for certain products and services in real time. This ability to recognize demand in real time, together with highly automated and flexible production environments, allow for the real-time fulfillment

of this demand. In an autonomous pull economy, the majority of labor will be carried out entirely by machines. Where machines cannot perform tasks completely autonomously, they will heavily augment human labor. As a result, humans will predominantly work in white-collar jobs overseeing machine labor, instead of blue-collar jobs where they themselves have to do the work.

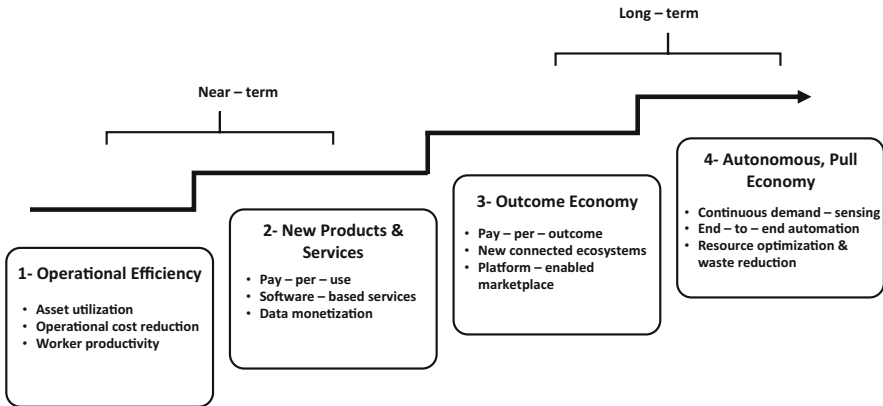


Fig. 10.8 Adoption and impact path of the Industrial Internet of Things (adapted from World Economic Forum (2015)).

10.3.4 *The Internet of Things in the Energy and Health Care Sectors*

As stated earlier, the Internet of Things will expectedly have a significant impact on many facets of people's daily lives, including their homes, cities, and work places. Two areas that are especially interesting and heavily affected by the Internet of Things are the energy and health care sectors. Since an in-depth discussion of all industry sectors that are possibly impacted by the Internet of Things is beyond the scope of this chapter, the remainder of this section, therefore, focuses on the energy and health care sectors to further illustrate the impact of the Internet of Things.

The Internet of Things in the Energy Sector

Although smart energy is certainly an integral aspect of smart homes and smart cities, its wider impact on the energy sector is probably most noticeable. The energy sector has readily embraced the Internet of Things with firms like Apache Corporation procuring predictive maintenance of oil pumps, turbines, and other machinery. More importantly, however, the Industrial Internet of Things opens up possibilities for saving significant amounts of energy, as well as delivering energy more efficiently (Bassi et al. 2013). Following incidents like the nuclear disasters at Chernobyl and Fukushima, as well as an increased awareness of the fossil fuels' environmental effects and dwindling availability, the energy sector is increasingly

shifting toward renewable energy sources. Furthermore, global demand for energy is steadily growing. However, the amount of energy that can be generated with renewable energy sources relies heavily on weather conditions and is, therefore, subject to stronger fluctuations than traditional energy sources. Smart grids present themselves as a solution to this dilemma, as well as other challenges in the energy sector. A smart grid is an electricity transmission and distribution system that uses ICTs to improve the electricity grid's reliability, security, and efficiency. Smart grids include decentralized, smart power plants that can detect and dynamically respond to changing energy demands, energy transmission networks that are capable of automatically optimizing overall energy flows, and smart homes that produce a part of the energy they require themselves. Moreover, smart appliances and smart meters are expected to help reduce the need for building new power plants (Bassi et al. 2013). Current grids are usually overprovisioned to withstand peak demands. Consequently, smart appliances can turn themselves off when not needed or turn themselves on during times of low power demands (e.g., a washing machine can automatically turn itself on at night), whereas smart meters enable customers to be aware of and reduce their power consumption. Overall, adopting the Internet of Things in the energy sector will cause significant changes in how energy is produced, distributed, and consumed.

The Internet of Things in the Health Care Sector

The healthcare industry, with its rapidly increasing data collection and shift towards a patient-centered precision medicine treatment paradigm (Thiebes et al. 2017a; Thiebes et al. 2017b), is another industry sector that will be heavily affected by the Internet of Things. Improving patients' health is the paramount goal in health care. Applying the Internet of Things in health care, which is occasionally also referred to as the medical Internet of Things or the Internet of Medical Things, offers several opportunities for improving patient care and patient health. The continuous and remote monitoring of the patients' physiological data represent a cornerstone of the medical Internet of Things. The proliferation of smart wearable devices, for example, has already resulted in an abundance of health-related data being recorded and being readily available to doctors (Spil et al. 2017). Today, most smart watches regularly measure their users' heart frequency. Furthermore, in a 2018 keynote, Apple Computer announced that their newest smart watch will even have the ability to conduct electrocardiograms. These smart watches also alert users if heartrate irregularities are detected. However, the possibilities of remote monitoring reach far beyond consumer-grade smart devices. Remote monitoring specifically includes professional medical devices, which are increasingly connected to the Internet to remotely provide doctors with the patients' vital signs and even to remotely give intensive care patients a dose of medicine, if required. Besides remote health monitoring, Ambient Assisted Living represents another important use case for the medical Internet of Things (Kromat et al. 2018; Wolf et al. 2017). With a rising number of senior citizens among the world population, the health systems in countries around the globe are facing major challenges (Fattah et al. 2017). Studies among senior citizens in the U.S. and Europe have shown that the majority of people

over 65 want to stay in their homes as long as possible (Pham et al. 2018). However, depending on their mental and physical conditions, being unsupervised at home can be dangerous for seniors and people with special needs. For example, seniors lying for a long time after a fall until help arrives find themselves in a very dangerous situation, which can not only cause discomfort, but lead to death within a few months (Yacchirema et al. 2018). Ambient Assisted Living encompasses technological systems comprised of sensors, smart objects, and smart devices that support senior citizens and people with special needs, thus enabling them to remain largely autonomous and continue living in their familiar environments (Dohr et al. 2010). Sensors could, for example, detect that a senior citizen has had a fall and accordingly alert an ambulance. In case people with dementia forget to turn off the stove, the stove could detect that it had been left switched on unsupervised and can then automatically turn itself off. The possibilities of the medical Internet of Things are manifold and this brief section only scratches the surface of what is already possible now and will be possible in the future. The medical Internet of Things provides many opportunities to improve health care around the world while simultaneously addressing challenges, such as aging populations and dwindling medical resources.

10.4 Challenges and the Future of the Internet of Things

10.4.1 Challenges

Positive aspects notwithstanding, the Internet of Things' increasing dissemination and adoption also brings forth several important challenges that need to be addressed such that this technology's full potential can be realized. These challenges are, among others (1) the flood of data created by an abundance of sensors, smart objects, and smart devices, (2) the interoperability of these technologies, as well as (3) the questions with regard to ensuring security and privacy in a world where everything can potentially be measured, monitored, and recorded, or hacked.

Data flood and the Internet of Things

A central characteristic of the Internet of Things is its ubiquity, realized by interspersing our physical world with interconnected sensors, smart objects, and smart devices. However, the rapidly increasing number of deployed sensors, smart objects, and smart devices is also accompanied by an unprecedented flood of data that needs to be transmitted, processed, and stored. While our current ICT infrastructures can still handle the amounts of produced data, several experts suggest that they will soon reach the limits of their capacity (daCosta 2013). Although the current Internet infrastructure is, in principle, overprovisioned for peak usage and can address an astonishing 2 to the power of 128 participants in the network, the requirements for realizing the Internet of Things differ entirely from our traditional Internet's requirements (daCosta 2013). In the current, traditional Internet, data is transferred in

relatively large chunks. When someone requests a website, for example, the request is answered by sending most of the website data and all the data required for displaying the website from a server to the requester. Moreover, the Internet is designed to guarantee the arrival of all requested data, as receiving only a part of a website or email would, for example, be highly impractical. In the Internet of Things, sensors and smart objects constantly send and/or receive small portions of information that need very little overhead throughout the day. As such, the majority of data will not be transferred in intervals of relatively large data chunks, but rather in a constant stream of small bits of data. Furthermore, in most cases it is not critical if, for example, a single measurement out of thousands of measurements per hour of a temperature sensor is lost, since a single temperature measurement within a specific point in time is much less important than many temperature measurements over a period of time (daCosta 2013). However, not only data transmissions within the age of the Internet of Things pose a serious challenge to our ICT infrastructures; storing and processing the produced data are also likely to require a rethink of our current ICT infrastructures. Within the current paradigm, most of the data produced by sensors, smart objects, and smart devices are sent to central cloud servers where the data are stored and processed, and, if necessary, used to initiate further action. However, even cloud services with their seemingly unlimited resources will sooner or later be overwhelmed by the vast amounts of data created by the Internet of Things. A possible solution to the problems of data transmission, storage, and processing in the age of the Internet of Things is moving data processing from the center of the Internet (i.e., central cloud services) closer to the points where the data had been produced (i.e., to its edges). This relatively new concept called edge computing will reduce the need for transferring extensive amounts of data to central cloud services, since the smart devices and smart objects will take over much of the network's and cloud services' load.

Interoperability

Interoperability is another crucial challenge that must be addressed in order for the Internet of Things to become a success story like the Internet. Take smart homes, for example. As described earlier, a classic smart home contains several smart objects and devices from a number of manufacturers, and all these objects and devices must communicate or interact effectively with each other to realize their benefits. People expect to control their IKEA or any other smart lights from their smart phones irrespective of whether they run Apple's iOS, Google's Android, or any other smart phone operating system. Currently, there are already a variety of competing standards and protocols for the Internet of Things, such as Z-Wave and Zigbee in the smart home cosmos, which were discussed in this chapter (see section 10.3.1). While many smart objects and devices currently support multiple standards and protocols, there is also a trend toward platformization and creating different ecosystems that do not cooperate well or at all among themselves. Interoperability becomes even more challenging when considering the Industrial Internet of Things' countless application scenarios among the diverse industries. Different industries will have different

objectives and requirements for their Industrial Internet of Things, requiring standards and protocols with a diverse set of capabilities. With the ongoing and accelerating diffusion of the Internet of Things, the number of available smart objects and devices, as well as the number of manufacturers and, with them, the number of competing standards, are likely to increase, further amplifying the problem of interoperability. A single standard or protocol that can fulfill all these varying and sometimes opposing needs, is unlikely. Nevertheless, a certain degree of interoperability is clearly necessary to realize the vision of a pervasive Internet of Things and its associated benefits.

Security and Privacy

Security and privacy are frequently discussed topics within the context of the Internet of Things. Although often used interchangeably, security and privacy – or more precisely, information security and information privacy – refer to two distinct concepts. Information security refers to protecting data against unauthorized access (e.g., by hackers) by ensuring confidentiality, integrity, and availability. Information privacy, instead, refers to an individual's ability to control who can access their data under which circumstances and for which purposes. Both security and privacy are clearly important concepts, considering the amount and kind of potentially private data the Internet of Things collects. For example, take a fully connected smart home with motion sensors to trigger lights in the hallway, a smart refrigerator that tracks the foods inside and automatically orders new foods online when necessary, and sensors to track who is at home at what time in order for smart thermostats to set the temperature accordingly. Or think about smart cities, where citizens' every move is tracked to detect crime or optimize traffic flows. While the intended purpose might be benevolent, all this collected information can potentially also be used unethically or even compromisingly to harm individuals and alter our societies unfavorably. The problem becomes even more evident when considering health-related information collected by smart watches and the medical Internet of Things. Not only could health-related information in the wrong hands cause harm to an individual's personal, social, and professional life (e.g., imagine a world in which your health is monitored 24 hours a day and your insurance policy is altered dynamically according to certain health metrics); it could also result in serious health consequences if hackers decide to hack and tamper with an individual's internet-connected medical devices. Consequently, security in the Internet of Things context always also includes a notion of safety (i.e., protecting humans, the environment, and machines against potentially harmful system failures) (Sadeghi et al. 2015). As a result of the significant security and privacy challenges associated with the Internet of Things' dissemination, researchers and practitioners likewise focus increasingly on developing secure and privacy-preserving concepts for the Internet of Things. Only if security is ensured and individuals are enabled to make informed decisions about who may use their data, when, and for what purpose, the Internet of Things will be able to fully unfold its potential to the benefit of everyone.

10.4.2 Outlook: The Future of the Internet of Things

Owing to the Internet of Things' rapid development and its underpinning technologies, it is still unclear how its impact on individuals and society will unfold in the future. Nevertheless, this last section of this chapter on the Internet of Things provides a brief overview of what can probably be expected from the Internet of Things in the future. Although the Internet of Things is one of the major topics among researchers in various disciplines (e.g., computer science, information systems, economics, medicine, ecology) and practitioners, and although an increasing amount of smart objects and smart devices is deployed, the Internet of Things is still in its early stages. According to the market research firm, Gartner, their famous hype cycle shows that the Internet of Things is currently moving from the peak of inflated expectations to the trough of disillusionment (Panetta 2018). Thus, the Internet of Things will only reach widespread mainstream adoption within the next five to ten years. Nevertheless, we are currently at an inflection point with regard to the Internet of Things. In the coming years, more physical things will become increasingly connected and people will notice an overall growth in the number of available smart objects and devices. Accordingly, current estimates suggest that we will experience up to 75 billion connected devices by the year 2025 (Statista 2019b). More cities will launch efforts and projects to become smart cities, people will increasingly notice semiautonomous cars and sooner or later even fully autonomous smart connected cars. Increasingly more industries will start to embrace the Internet of Things and especially the energy, health care, and manufacturing sectors will experience a deeper penetration of the Internet of Things and explore new means for better utilizing its possibilities. With the Internet of Things' increasing diffusion, new industry sectors will be created, and new job opportunities will arise. However, this will also result in new workforce requirements to which people will have to adapt. In the near future, we will probably experience more security incidents and privacy breaches that are related to the Internet of Things. These incidents and breaches will doubtlessly focus the discussion about the Internet of Things on potential security and privacy issues and lead to more efforts for securing the Internet of Things.

Fully-connected smart homes are still costly and an exception rather than the norm, and we must still see a real smart city at scale. Moreover, many open questions with regard to the Industrial Internet of Things remain, with ideas like a fully automated, Internet of Things-based pull economy still years away from becoming reality. Only time can tell, if, in the long run, the Internet of Things can live up to the hype surrounding it and successfully move from the trough of disillusionment to the plateau of productivity. The coming years will certainly be eventful and exciting for the Internet of Things.

Summary

The Internet of Things is one of the major technology trends of the last few years. The term "Internet of Things" was coined in the early 1990s to describe a paradigm in which not only human-to-human and human-to-machine communication, but also machine-to-machine communication that occurs over the Internet, are ubiquitous. The concept is characterized by a self-configuring, adaptive, complex network that interconnects things, which possess a physical and virtual representation over the Internet by means of standard communication protocols. As such, the Internet of Things' nine central characteristics comprise (1) the interconnection of things (i.e., any physical object that is relevant to users of the Internet of Things), (2) the connection of these things to the Internet, (3) the ability to uniquely identify things, (4) ubiquity of the network, (5) sensing and actuation capabilities, (6) embedded intelligence, (7) interoperable communication capabilities, (8) self-configurability, and (9) programmability.

Besides the Internet, which serves as the Internet of Things' backbone, four categories of technologies can be differentiated that together serve as enablers for realizing the Internet of Things. First, the large number of potential objects that ought to be tracked in an Internet of Things requires inexpensive and easy-to-use tagging technologies. RFID is a tagging technology that sets out to address these requirements and is closely connected to the idea of the Internet of Things. Second, sensing is a core capability of the Internet of Things that is driven by the rapidly decreasing costs of all kinds of sensors. Sensors come in various sizes and form factors, can be wired or wireless, and exhibit a wide range of sensing capabilities. Third, smart technologies, such as smart objects (i.e., physical objects with the ability to recognize, process, and exchange data, act autonomously, adapt flexibly to specific situations, and interact with humans or other smart technologies) and smart devices (i.e., portable multi-purposes ICT devices that enable access to several application services located locally on the device or remotely on servers), which together form smart environments, are required to process the vast amounts of data that are being created by the Internet of Things. Fourth, in order to unobtrusively embed intelligence into almost all kinds of everyday physical objects, advanced miniaturization technologies are necessary to make computer chips and sensors increasingly smaller. Over the last few years, researchers and practitioners have proposed several architectural models for the Internet of Things. The two most basic types of architectural models for the Internet of Things are the three-layer model with a perception layer at the bottom, an application layer at the top, and a network layer in between the perception and application layers, and the five-layer model, which includes a perception layer at the very bottom, followed by a transport layer, a processing layer, an application layer, and the business layer at the very top.

Important application areas for the Internet of Things are, among others, smart homes, smart cities, and the Industrial Internet of Things. Smart homes are homes that are interspersed with smart objects and smart devices, giving them awareness about what happens inside and the ability to act upon their residents' commands or

autonomously in case of specific events. Smart homes strive to improve living comfort, while simultaneously decreasing resource usage and increasing security. Smart speakers driven by artificial intelligences like Alexa, Siri, or Google Assistant often serve as central, voice-controlled, control centers for smart homes. In contrast to smart homes, smart cities are sustainably developed urban areas with a high quality of living. They outperform non-smart cities in the areas of economy, people, governance, mobility, environment, and living through a combination of strong human and social capital, as well as a modern and pervasive ICT infrastructure. The Industrial Internet of Things involves the application of Internet of Things technologies (e.g., smart objects and smart devices) within an industrial setting in order to promote goals that are unique to industry. The Industrial Internet of Things includes diverse industries, such as manufacturing, agriculture, retail, logistics, and aviation. Two industries that are specifically affected by the Internet of Things' emergence are the energy and health care sectors.

Finally, the Internet of Things' increasing dissemination and adoption also brings forth several important challenges that need to be addressed in order for this technology to realize its full potential. Among these challenges are the flood of data that is being created by an abundance of sensors, smart objects, and smart devices, the interoperability of these technologies, as well as questions related to ensuring security and privacy in a world where everything can potentially be measured, monitored, and recorded.

Questions

1. What is the Internet of Things and what are its nine essential characteristics?
2. Who coined the term "Internet of Things" and when?
3. When did the Internet of Things come into existence?
4. What are the four enabling technologies for the Internet of Things and how do they enable the Internet of Things?
5. What are the three core concepts of the Internet of Things?
6. What are the standard application areas of the Internet of Things?
7. What are the central challenges for the Internet of Things?

References

- Ashton K (2009) That 'internet of things' thing. *RFID J* 22(7):97–114
- Bassi A, Bauer M, Fiedler M, van Kranenburg R, Kramp T, Lange S, Meissner S (2013) Enabling things to talk: designing IoT solutions with the IoT architectural reference model. Springer, Heidelberg
- Benight SJ, Wang C, Tok JBH, Bao Z (2013) Stretchable and self-healing polymers and devices for electronic skin. *Prog Polym Sci* 38(12):1961–1977

- Boyes H, Hallaq B, Cunningham J, Watson T (2018) The industrial internet of things (IIoT): an analysis framework. *Comput Ind* 101:1–12
- Carrez F, Bauer M, Boussad M, Bui N, Jardak C, De Loof J, Magerkurth C, Meissner S, Netzsträter A, Oliveureau A (2013) Internet of things—architecture IoT-A, deliverable D1. 5—final architectural reference model for the IoT v3. 0. https://www.researchgate.net/profile/Walewski_Joachim/publication/272814818_Internet_of_Things_-_Architecture_IoT-A_Deliverable_D15_-_Final_architectural_reference_model_for_the_IoT_v30/links/58f78bf00f7e9bfc93bde4c/Internet-of-Things-Architecture-IoT-A-Deliverable-D15-Final-architectural-reference-model-for-the-IoT-v30.pdf. Accessed 17 Sept 2019
- Chen Y (2018) Research on the connotation and business model of internet of things under the background of globalization. *Proc Bus Econ Stud* 1(2):16–20
- Chen J, Walz E, Lafferty B, McReynolds J, Green K, Ray J, Mulvenon J (2018) China's internet of things. https://www.uscc.gov/sites/default/files/Research/SOSi_China%27s%20Internet%20of%20Things.pdf. Accessed 21 Apr 2019
- Cisco (2014) The internet of things reference model. http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf. Accessed 16 Jan 2019
- daCosta F (2013) Rethinking the internet of things: a scalable approach to connecting everything, 1st edn. Apress, Berkeley, CA
- Daugherty P, Banerjee P, Negm W, Alter AE (2015) Driving unconventional growth through the industrial internet of things. Accenture. https://www.accenture.com/ph-en/_acnmedia/Accenture/next-gen/reassembling-industry/pdf/Accenture-Driving-Unconventional-Growth-through-IIoT.pdf. Accessed 16 Jan 2019
- Department of Transportation (2016) Smart city challenge: lessons for building cities of the future. <https://www.transportation.gov/sites/dot.gov/files/docs/Smart%20City%20Challenge%20Lessons%20Learned.pdf>. Accessed 17 Sept 2019
- Dohr A, Modre-Oprian R, Drobits M, Hayn D, Schreier G (2010) The internet of things for ambient assisted living. Paper presented at the 7th international conference on information technology, Las Vegas, NV, 12–14 Apr 2010
- Fattah S, Sung N-M, Ahn I-Y, Ryu M, Yun J (2017) Building IoT services for aging in place using standard-based IoT platforms and heterogeneous IoT products. *Sensors* 17(10):2311–2340
- Gartner (2014) Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business. <https://www.gartner.com/en/newsroom/press-releases/2014-08-11-gartners-2014-hype-cycle-for-emerging-technologies-maps-the-journey-to-digital-business>. Accessed 17 Sept 2019
- Giffinger R (2007) Smart cities. Ranking of European medium-sized cities. http://www.smart-cities.eu/download/smart_cities_final_report.pdf. Accessed 2 Apr 2019
- Gilchrist A (2016) Industry 4.0: the industrial internet of things, 1st edn. Apress, Berkeley, CA
- International Telecommunication Union (2005) ITU internet reports 2005: the internet of things. <https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf>. Accessed 5 Jan 2019
- Klewes J, Popp D, Rost-Hein M (2017) Out-thinking organizational communications: the impact of digital transformation. Springer, Geneva
- Kortuem G, Kawsar F, Sundramoorthy V, Fitton D (2009) Smart objects as building blocks for the internet of things. *IEEE Internet Comput* 14(1):44–51
- Kromat T, Dehling T, Haux R, Peters C, Sick B, Tomforde S, Wolf K-H, Sunyaev A (2018) Gestaltungsraum für proaktive Smart Homes zur Gesundheitsförderung. Paper presented at the Multikonferenz Wirtschaftsinformatik, Lüneburg, 7 Mar 2018
- López TS, Ranasinghe DC, Patkai B, McFarlane D (2011) Taxonomy, technology and applications of smart objects. *Inf Syst Front* 13(2):281–300
- Lucke D, Constantinescu C, Westkämper E (2008) Smart factory - a step towards the next generation of manufacturing. Paper presented at the 41st CIRP conference on manufacturing systems, Tokyo, 26–28 May 2008

- McKerracher C, Torriti J (2013) Energy consumption feedback in perspective: integrating Australian data to meta-analyses on in-home displays. *Energy Effic* 6(2):387–405
- Minerva R, Biru A, Rotondi D (2015) Towards a definition of the internet of things (IoT). https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf. Accessed 17 Jan 2019
- Miorandi D, Sicari S, De Pellegrini F, Chlamtac I (2012) Internet of things: vision, applications and research challenges. *Ad Hoc Netw* 10(7):1497–1516
- Misra J (2017) IoT system I sensors and actuators. <https://bridgera.com/iot-system-sensors-actuators/>. Accessed 13 Mar 2019
- Muntjir M, Rahul M, Alhumiany H (2017) An analysis of internet of things (IoT): novel architectures, modern applications, security aspects and future scope with latest case studies. *Build Serv Eng Res Technol* 6(6):422–448
- Ove Arup & Partners (2013) The smart city market: opportunities for the UK. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/249423/bis-13-1217-smart-city-market-opportunities-uk.pdf. Accessed 17 Sept 2019
- Panetta K (2018) Widespread artificial intelligence, biohacking, new platforms and immersive experiences dominate this year's Gartner Hype Cycle. <https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/>. Accessed 16 Apr 2019
- Pham M, Mengistu Y, Do H, Sheng W (2018) Delivering home healthcare through a cloud-based smart home environment (CoSHE). *Futur Gener Comput Syst* 81:129–140
- Poslad S (2011) Ubiquitous computing: smart devices, environments and interactions. Wiley, Chichester
- Roland Berger (2018) Strategie der Digitalstadt Darmstadt. <https://dabei.digitalstadt-darmstadt.de/digitalstadt/de/home/file/fileId/214/name/Strategie%20der%20Digitalstadt%20Darmstadt>. Accessed 17 Sept 2019
- Romkey J (2017) Toast of the IoT: the 1990 interop internet toaster. *IEEE Consum Electron Mag* 6(1):116–119
- Sadeghi A-R, Wachsmann C, Waidner M (2015) Security and privacy challenges in industrial internet of things. Paper presented at the 52nd ACM/EDAC/IEEE design automation conference, San Francisco, CA, 8–12 June 2015
- SAP SE (2018) SAP® Leonardo internet of things. Geschäftserfolg in einer vernetzten Welt. https://www.t-h.de/fileadmin/content/downloads/whitepaper/SAP_Leonardo_Internet_of_Things_-_Geschaeftserfolg_in_einer_vernetzten_Welt.pdf. Accessed 21 Apr 2019
- Schoder D (2018) Introduction to the internet of things. In: Hassan QF (ed) Introduction to the internet of things. Internet of things A to Z: technologies and applications. Wiley, Hoboken, NJ, pp 3–50
- Sethi P, Sarangi SR (2017) Internet of things: architectures, protocols, and applications. *J Electr Comput Eng* 2017(1):1–25
- Spil T, Sunyaev A, Thiebes S, Van Baalen R (2017) The adoption of wearables for a healthy lifestyle: can gamification help? Paper presented at the 50th Hawaii international conference on system sciences, Waikoloa Village, HI, 4–7 Jan 2017
- Statista (2019a) Global digital population as of January 2019 (in millions). <https://www.statista.com/statistics/617136/digital-population-worldwide/>. Accessed 13 Apr 2019
- Statista (2019b) Internet of things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>. Accessed 13 Apr 2019
- Statista (2019c) Percentage of households with internet access worldwide in 2018, by region. <https://www.statista.com/statistics/249830/households-with-internet-access-worldwide-by-region/>. Accessed 13 Apr 2019
- Strehmann J (2017) Bitkom-Wettbewerb: Digitale Stadt. <https://www.digitalestadt.org/de/bitkom/org/Digitale-Stadt/Wettbewerb>. Accessed 5 Feb 2019

- Swan M (2012) Sensor mania! The internet of things, wearable computing, objective metrics, and the quantified self 2.0. *J Sens Actuator Netw* 1(3):217–253
- Thiebes S, Kleiber G, Sunyaev A (2017a) Cancer genomics research in the cloud: a taxonomy of genome data sets. Paper presented at the 4th international workshop on genome privacy and security, Orlando, FL, 15 Oct 2017
- Thiebes S, Lyytinen K, Sunyaev A (2017b) Sharing is about caring? Motivating and discouraging factors in sharing individual genomic data. Paper presented at the 38th international conference on information systems, Seoul, 10–13 Dec 2017
- Verdone R, Dardari D, Mazzini G, Conti A (2010) *Wireless sensor and actuator networks: technologies, analysis and design*. Academic Press, London
- Warneke B, Last M, Liebowitz B, Pister KS (2001) Smart dust: communicating with a cubic-millimeter computer. *Computer* 34(1):44–51
- Weiser M (1991) The computer for the 21st century. *Sci Am* 265(3):94–105
- Wolf K-H, Dehling T, Haux R, Sick B, Sunyaev A, Tomforde S (2017) On methodological and technological challenges for proactive health management in smart homes. *Stud Health Technol Inform* 238:209–212
- World Economic Forum (2015) Industrial internet of things: unleashing the potential of connected products and services. http://www3.weforum.org/docs/WEFUSA_IndustrialInternet_Report2015.pdf. Accessed 10 Mar 2019
- Yacchirema D, de Puga JS, Palau C, Esteve M (2018) Fall detection system for elderly people using IoT and big data. *Procedia Comput Sci* 130:603–610

Further Reading

- Bassi A, Bauer M, Fiedler M, van Kranenburg R, Kramp T, Lange S, Meissner S (2013) *Enabling things to talk: designing IoT solutions with the IoT architectural reference model*. Springer, Heidelberg
- Carrez F, Bauer M, Boussad M, Bui N, Jardak C, De Loof J, Magerkurth C, Meissner S, Nettsträter A, Olivereau A (2013) Internet of things—architecture IoT-A, deliverable D1. 5—final architectural reference model for the IoT v3. 0. https://www.researchgate.net/profile/Walewski_Joachim/publication/272814818_Internet_of_Things_-_Architecture_IoT-A_Deliverable_D15_-_Final_architectural_reference_model_for_the_IoT_v30/links/58f78bf00f7e9bfcf93bde4c/Internet-of-Things-Architecture-IoT-A-Deliverable-D15-Final-architectural-reference-model-for-the-IoT-v30.pdf. Accessed 17 Sept 2019
- Cisco (2014) The internet of things reference model. http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf. Accessed 16 Jan 2019
- daCosta F (2013) *Rethinking the internet of things: a scalable approach to connecting everything*, 1st edn. Apress, Berkeley, CA
- Gilchrist A (2016) *Industry 4.0: the industrial internet of things*, 1st edn. Apress, Berkeley, CA
- Hassan QF (2018) *Introduction to the internet of things. Internet of things A to Z: technologies and applications*. Wiley, Hoboken, NJ

Chapter 11

Critical Information Infrastructures



Abstract

Information systems have evolved rapidly in the past decades and increasingly take a central role in society. Today, some information systems have become such integral parts of society that their disruption or unintended consequences can have detrimental effects on vital societal functions; that is, they have become critical information infrastructures. This chapter clarifies the concept of ‘critical information infrastructures’ and distinguishes them from conventional critical infrastructures. After introducing foundational concepts and the evolution of information infrastructures, the chapter discusses salient characteristics, important challenges, main functions, and core tasks for operating critical information infrastructures. Critical information infrastructures, in spite of their vital role in society, often go unnoticed. In this chapter, the reader learns the basics of recognizing, understanding, and operating critical information infrastructures.

Learning Objectives of this Chapter

This chapter has five main learning objectives. First, readers should become familiar with the evolution of information infrastructures and get to know where they started. Second, readers should understand the complex nature of critical information infrastructures and learn how to analyze and design such systems. Third, readers should be able to recognize and distinguish between critical infrastructures and critical information infrastructures. Fourth, readers should understand the main properties and functions of critical information infrastructures and be enabled to manage them in practice. Last, but not least, readers should become acquainted with the key challenges they will encounter during the operation of critical information infrastructures and become properly equipped to master these challenges when they encounter them in the field.

Structure of this Chapter

Section 11.1 addresses the foundations of critical information infrastructures by outlining the technological transformations relevant to grasping the concept of critical information infrastructures, introducing the notions of sociotechnical and sociomaterial information systems, presenting a conceptualization of critical information infrastructures, and discussing the commonalities and differences between

critical infrastructures and critical information infrastructures. Section 11.2 introduces the reader to the main properties of critical information infrastructures. Section 11.3 presents a typology of the main functions of critical information infrastructures and illustrates them with real-world examples. Section 11.4 covers the operation of critical information infrastructures, and concludes the chapter with a discussion of the key challenges that readers would have to master to successfully design and operate critical information infrastructures.

11.1 Foundations of Critical Information Infrastructures

Information technology has developed rapidly in the last couple of decades. Mainframes have been replaced by personal computers and mobile devices. Moreover, the Internet has enabled the interconnection of almost any device on a global scale. Information technology has permeated nearly all parts of the economy, society, and daily life. People have become more and more dependent on information systems. Information technology is used, for instance, to control airports and public transportation, to coordinate logistic flows, and to manage patient care (Dehling et al. 2015). Important societal functions served by information systems include providing health care, safety and security, globally interconnected supply chains, and social and economic welfare. Information systems capable of causing conditions detrimental to vital societal functions constitute critical information infrastructures (CIIs). Managing the complexity of CIIs, understanding their functioning, and grasping their impact requires a thorough understanding of their foundations, which will be discussed in this chapter.

11.1.1 *The Emergence of Critical Information Infrastructures*

With the emergence of CIIs, society has become more dependent on information technology than ever before. Recent history can illustrate technological transformations that have triggered radical economic and societal change. A look at these technological transformations is helpful for better understanding the emergence of CIIs and how they can impact and transform important societal service structures.

Radical change in economies and societies is often driven by technological transformation (Ayres 1990). A first transformation took place between 1770 and 1800 in the UK and some parts of Europe, when transport networks were scaled up, facilitating a shift to large-scale use of coal for energy provision through steam engines. By 1830, railways started to replace canals for transportation of heavy goods, and gas lighting and telegraph networks were introduced. During the third transformation between 1870 and 1895, the invention of internal combustion

engines facilitated the emergence of automobiles, which resulted in the appearance of new industries, such as the petroleum and the electrical industries. After World War II, a rising consumer demand created growth in already established industries and led to the introduction of many new products and services, such as consumer electronics and air transportation. From the 1970s onwards, a fifth transformation paved the way for CIIs, namely, the emergence and increasing dissemination of telecommunication and computer technologies.

By that time, computer technologies had already been developing for quite a while. In 1890, Herman Hollerith designed a punch card system and established a company that ultimately became the current IT company IBM. In 1936, Alan Turing presented his Turing machine, which became the central concept of the modern computer. On March 31, 1951, the first commercial computer, the UNIVAC I (UNIVersal Automatic Computer I), was delivered. In 1953, Grace Hopper developed the first programming language (COBOL). In 1973, the first links from the European Union to the US-American ARPANET, the predecessor of the Internet, were established. The first computer with a single circuit board was the Apple I, which was rolled out in 1976. In 1981, IBM presented the first personal computer, which used Microsoft's MS-DOS operating system. In 1985, Microsoft announced its Windows operating system and the first dot-com domain name was registered. In 1993, the first Pentium microprocessor enhanced the use of graphics and music on personal computers. The Google web search engine was developed by Sergey Brin and Larry Page in 1996. In 1999, the term Wi-Fi made it into the computer language. In 2003, AMD made the first 64-bit processors available to the consumer market. The Web 2.0 gained traction in 2004, representing a move away from unilateral communication from providers to users, toward more interaction between users. This was accompanied by the launch of social networking services, such as Facebook. In 2006, the Web 3.0 represented the next evolution of the Internet and leveraged semantic markup and web services to reuse more data across different applications and services. The appearance of smartphones in 2007 constituted the beginning of the Web 4.0 and increased use of mobile and ubiquitous devices on the Internet.

Telecommunications and computer technologies evolved slowly, though sporadically, they exhibited by rapid leaps (e.g., transistors, smartphones). The past few decades have seen an ever-increasing speed of evolution, to the point that computers are now widely used by the general public. Currently, many major companies build their business models around computer technologies and the Internet (e.g., web search engines, social networking services, or online retailers) and governments are digitalizing public policy processes. In almost all aspects of daily life users have become increasingly dependent on telecommunication and computer technologies.

The way in which information technology permeates everyday activities is closely connected to the emergence of CIIs. With rising maturity of telecommunication and computer technologies, more and more activities in businesses, management, and other areas of daily life have become dependent on information technology. This means that CIIs not only constitute technological innovations but are also deeply ingrained in social processes. Accordingly, understanding CIIs

requires that we look at the technical and social components of CIIs. The next section outlines central aspects of sociotechnical systems.

11.1.2 Sociotechnical Systems

Sociotechnical systems, and the sociotechnical perspective in particular, have been subject to intensive researched, even before rapid digitalization. A prominent example of the interplay between technical and societal perspectives comes from the coal mining industry of the 1950s in the UK (Trist 1981), illustrating how technology affects social structures in a way that, even today, still helps us grasp the importance of the sociotechnical perspective in the digital age.

The UK coal mining industry introduced new technology in the mines in the 1950s to increase mechanization and create better working conditions. However, the introduction of new mining technology backfired in that it led to decreased productivity, high absenteeism, and staff losses. In short, mining management had ignored the sociotechnical perspective.

The traditional mining process was known as shortwall mining. Coal was mined in small self-organized groups that worked in parallel tunnels dug horizontally into the coal seam. These groups were relatively autonomous in their decision-making; miners were skilled to perform diverse roles as circumstances required, and they were paid based on the overall group output. Increased mechanization led to longwall mining, which meant that coal was excavated vertically along the front of the coal seam and carted to the surface by conveyor belts. Workers were assigned to single tasks and were mostly skilled to perform only one role. Their supervision was externalized, and they were paid on the basis of how well they performed in this role.

When longwall mining was introduced mine management's focus was on increasing mechanization, and they did not foresee the range of problems this would entail. Everyday operations in mining are frequently interrupted by unexpected events. The streamlined processes and the specialized workforce made it hard to handle unexpected events efficiently. Furthermore, the mine layout brought about longer distances between the workforce and supervisors. In contrast to supervising in a factory, it is difficult to oversee operations in narrow, evolving, underground mine shafts. With the changes, workers became less motivated and gave less cooperation in response to their simpler, less flexible jobs. The new foundation for payment scheme disincentivized a focus on the overall mining output, narrowing each worker's attention to their own tasks only. Overall, this led to decreased productivity despite the introduction of new technology which was meant to increase productivity.

The example from a mining context shows that mere introduction of new technology is not enough to improve work processes. Instead, social and technical components must be well aligned and jointly improved to reap rewards of introducing new technology in contexts where technology becomes an integral part of social processes. Technical components that should have been considered in more detail in

the mining example described above, are the required level of mechanization, the scope of individual job descriptions, and the design of the overall production process. Social components that should have gotten more attention are the resulting reorganization of occupational roles and structures, the changes to the foundations for payment, the changed supervisory relationships, and the overall work culture.

CII, and information systems in general, are inherently sociotechnical, so that a sociotechnical perspective cannot be ignored. Key components that have to be considered when managing or changing CII are the technical subsystems with their physical systems and tasks, and the social subsystems with their structures and people (Bostrom and Heinen 1977). Fig. 11.1 depicts the main components of sociotechnical systems and their interdependencies. Relevant levels of analysis for sociotechnical systems are primary systems, the information system as a whole, and the macrosocial system (Trist 1981). Primary systems are subsystems with a specific set of activities operating within a larger information system (e.g., a database). A focus on a CII in its entirety (e.g., an air traffic control system) corresponds to a level of analysis that looks at the whole information system. A study of information systems on the macrosocial-system level considers how information systems operate within society.

A perspective that goes beyond the sociotechnical one is a sociomaterial perspective. With information technology deeply integrated in everyday activities, social and technical systems create entirely new effects through their constitutive entanglement (Orlikowski 2007). Such effects are studied through the lens of sociomateriality (Orlikowski and Scott 2008).

Orlikowski exemplified sociomateriality within the context of information search (Orlikowski 2007). The Google web search engine was initially a purely technical project aimed at indexing the world wide web and facilitating information retrieval. With its rising popularity, the Google web search engine served not only as the foundation for a large IT company; it became part of everyday life to the extent that it made its way into everyday language, where the verb ‘to google’ was generated as a shorthand equivalent to ‘searching information on the Internet.’ However, the Google web search engine is much more deeply ingrained in society and its services are dependent on technical and social aspects. The inner workings of Google’s web search engine constitute a useful example to illustrate this impact.

The search results retrieved from Google depend not only on a fit with the issued search query but also on the page rank of fitting websites. Websites are more likely to be part of the results if they have more links to other websites. In addition, the links’ weight is determined by the status of linked websites; for instance, official websites of some institutions are weighted higher than other sites mentioning the institution. Hence, Google search results depend on the algorithms that have been entered, on the proper functioning of the underlying hardware, and on the links that people set between their websites. Accordingly, Google search results are dynamic and evolve as new associations are made by people on the Internet. Hence, they emerge through entanglement of the code Google engineers produce and users’ actions on the Internet.

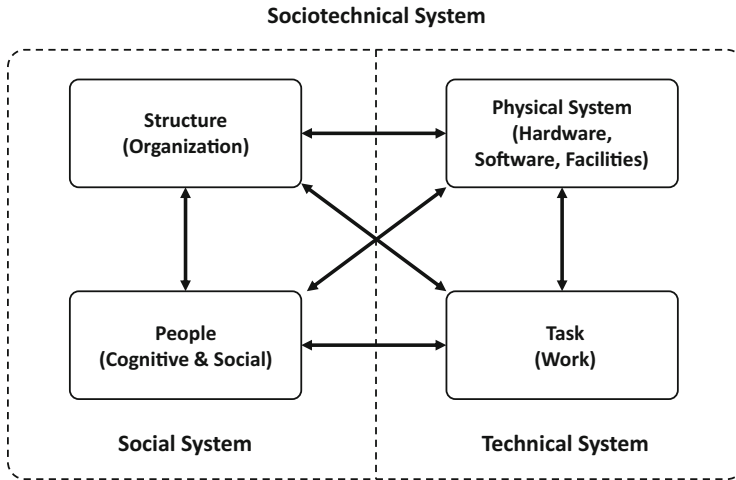


Fig. 11.1 Main social and technical components in critical information infrastructures (adapted from Bostrom and Heinen (1977))

Such entanglement of the Internet's social and technical properties is not without risk. For instance, with increased personalization on the Internet, the challenges regarding the sociomaterial nature of information search have intensified. One particular problematic effect is the creation of filter bubbles (Carroll 2017). Filter bubbles refer to search results that are tailored to the information that a provider has of a user. Hence, users are only provided with information they already know or like and not with information that might alienate them. When there are filter bubbles, users have limited opportunity to learn something new or be confronted with conflicting world views. Content producers exacerbate the impact of filter bubbles by rephrasing or changing their content in ways deemed suitable for as many different communities as possible (Carroll 2017). Consequently, some aspects may be less represented on the Internet because they are only of interest to a limited audience, and others are overrepresented because they are popular with many people. This means that user preferences are shaped by filter bubbles in contexts ranging from relatively benign (e.g., online shopping preferences) to critical (e.g., voting preferences). Filter bubbles may spare users from being confronted with irrelevant information, but, more importantly, they create new avenues for manipulation and (self-)propaganda. Hence, web search engines can be considered a CII because they can have a severe impact on information dissemination within a society. Web search engines impact what information people are likely to find; on the flip side, they also impact what information becomes very difficult to find. The next section offers a more detailed look at the nature of CII.

11.1.3 *Conceptualization of Critical Information Infrastructures*

CIIIs are considered to be related to critical infrastructures. However, unlike critical infrastructures, CIIIs focus on information processing and flows, not on physical infrastructure per se. Critical infrastructures, such as communication networks, are of marginal importance in the area of CIIIs. Instead, they constitute the necessary environment for the creation and functioning of CIIIs. A CII can be defined as *an information system that, if it is disrupted or has unintended consequences, can have detrimental effects on vital societal functions or the health, safety, security, or economic and social well-being of people* Adapted from (Union 2008).

Critical Information Infrastructure

An information system that, if it is disrupted or has unintended consequences, can have detrimental effects on vital societal functions or the health, safety, security, or economic and social well-being of people (Union 2008).

Information infrastructures support the creation, retrieval, processing, dissemination, and removal of information. Information infrastructures are considered critical if their failure can have consequences of critical proportions in magnitude, the breadth of its effect, and the extent of its duration (Egan 2007; Fekete 2011). Magnitude is assessed in terms of direct human damage (e.g., injuries, fatalities), economic loss (e.g., damage to whole industries), market failure (e.g., stock market crashes), damage to public infrastructure (e.g., failure of emergency services), or damage on a societal level (e.g., election manipulation). The breadth of effect is assessed on the basis of how many people, countries, or dependent critical infrastructures will be immediately affected by the failure of a CII. Similarly, the duration of how long a CII's disruption or its consequences last, and the time needed to repair it to full operating capability, can be extensive and, therefore, critical.

We find an example of a CII that emerged almost unnoticed in the unprotected Internet of Things (IoT) devices that were taken over by the Mirai computer worm that started infecting IoT devices on August 4, 2016. In the first 20 hours, almost 65,000 IoT devices were infected and the resulting botnets quickly reached stable dimensions of 200,000 to 300,000 IoT devices, with a peak of 600,000 IoT devices (Antonakakis et al. 2017). Most of the infected IoT devices were concentrated in South America and Southeast Asia. The Mirai botnets were mainly used to carry out distributed denial of service (DDoS) attacks: Between September 2016 and February 2017, they carried out just over 15,000 such attacks (Antonakakis et al. 2017) that were focused mainly on targets in the USA, but France and the UK were also heavily targeted. On October 21, 2016, there was a serious attack on the Domain Name System (DNS) provider Dyn (Antonakakis et al. 2017), as a result of which large sites such as Amazon, Netflix, PayPal, and Twitter were not available to customers for hours. A surprising aspect of Mirai was that it created botnets that could run

massive DDoS attacks even on low-powered IoT devices with simple dictionary attacks using standard passwords.

The Mirai example shows how CII's can develop and hardly be noticed as vital, until negative consequences appear. Neglect of security and the widespread use of default passwords, meant a large number of IoT devices were virtually unsecured. With limited criminal ingenuity, these IoT devices could easily be integrated into Mirai botnets. As a result, websites around the world had to endure and manage massive DDoS attacks for months. It is unlikely that IoT vendors intended to create a CII with the ability to dismantle the main websites around the world, but their lack of attention to security contributed significantly to the creation of exactly such a CII. The Mirai example also illustrates how, in contrast to critical infrastructures, some CII's are created on benign devices that are then exploited for malicious purposes on a large scale. IoT devices by themselves are not a CII, but the Mirai computer worm exploited them and turned them into a CII that produced unintended consequences that adversely affected major online players. The next section discusses the differences between critical infrastructures and CII's in more detail.

11.1.4 Differences Between Critical Infrastructures and Critical Information Infrastructures

A critical infrastructure can be defined as *a system that is essential for the maintenance of vital societal functions or the health, safety, or economic and social well-being of people* (Union 2008). Critical infrastructures are, for instance, oil supply, electric power grids, water supply, transportation networks, gas supply, and telecommunication networks.

Critical Infrastructure

A system that is essential for the maintenance of vital societal functions or the health, safety, or economic and social well-being of people (Union 2008).

The eruption of the Icelandic volcano Eyjafjallajökull, which was first noticed on February 26, 2010, provides a prominent example of the critical nature of the air transportation network. The volcano was covered by an ice cap which initially chilled the rising lava and turned it into small ash particles. Once the melted water came into contact with the lava, the resulting steam created an ash plume that propelled ash directly into the Northern Polar Jet Stream, which carried the ash to Western and Northern Europe. Since the ash could damage plane engines, almost all of the European airspace was closed for a period of six days. Within eight days, the eruption resulted in 107,000 flights being cancelled, which amounted to 48% of global air traffic at that time and a loss of \$1.7 billion for the airline industry (Bye 2011). The volcanic eruption had the effect of around 10 million passengers being stranded all over the world.

Critical infrastructures have a broader perspective than CIIs. CIIs are more focused, and they are concerned with information processing and flow within infrastructures. Literally, critical infrastructures do not seem very different to CIIs, but there are a few key characteristics with respect to design or operation on which critical infrastructures differ from CIIs. See Fig. 11.2 for an overview.

Critical infrastructures differ from CIIs on five design characteristics. First, critical infrastructures and CIIs differ as to their structural nature. Critical infrastructures (e.g., the electrical grid) are more tangible than CIIs in that they predominantly comprise physical objects, such as pipes, streets, cables, and buildings. CIIs, in contrast, predominantly comprise immaterial objects, such as software logic, applications, information.

Second, critical infrastructure and CIIs leverage platforms with different main building blocks. The focal components of critical infrastructures are hardware components, whereas the focal components of CIIs are software components.

Third, critical infrastructure and CIIs have different governance models. Critical infrastructures are usually purposefully developed to fulfill functions of important societal value. Accordingly, they are usually managed and controlled by public sector institutions. CIIs, on the other hand, become critical over time and are usually not designed to be critical from the outset. Accordingly, they are usually controlled by the private sector entities that initially deployed them. An example of a CII that is controlled by a private sector entity is the Facebook social networking service. Social networking services are not considered critical because of detrimental effects when they are disrupted. Rather, they are critical because they can produce unintended consequences with detrimental effects. This, for example, has been demonstrated by the potential for widespread manipulation of public opinion as was exhibited by the Cambridge Analytica incident, where Facebook's support of third-party apps was exploited to extract huge amounts of personal data from millions of Facebook users (Cadwalladr and Graham-Harrison 2018). Since the Cambridge Analytica incident, there has been increasing pressure for stronger regulation of social networking services. Even so, the Facebook social networking service is still controlled by the Facebook corporation.

Fourth, people engage differently with critical infrastructures and CIIs. While people typically benefit from critical infrastructures, they are usually not in direct contact with them. In CIIs, people often interact directly with the CII. Hence, CIIs are more visible to end-users than critical infrastructures are.

Fifth, data plays a different role in critical infrastructures and CIIs. In today's digitalized world, both critical infrastructures and CIIs generate data and leverage information technology for their operation and management. However, for controlling physical infrastructure, data is a source of performance optimization, while for CIIs data is essential due to how they control information flows. Thus, critical infrastructures have other components that are more essential than data. For example, a city's sewage system predominantly requires sewers and drains to lead waste water away; data plays only a minor role. In contrast, a social networking service like Facebook would cease to exist without data.

Dimension	 Nature	 Platform	 Governance	 People	 Data	 Initial Cost	 Deterioration	 Evolution	 Consequences
Critical Infrastructures	Tangible	Hardware	Public	Beneficiaries	Optional	Expensive	Constant	Slow	Assessable
Critical Information Infrastructures	Intangible	Software	Private	Direct Interaction	Essential	Negligible	None	Rapid	Undeterminable

differences in design characteristics

differences in operational characteristics

Fig. 11.2 Differences between critical infrastructures and critical information infrastructures.

In addition to the differences in design characteristics, critical infrastructures and CIIs differ on four operational characteristics. First, critical infrastructures and CIIs differ as to their initial cost. Critical infrastructures are often explicitly designed to serve purposes of critical breadth and critical duration. Therefore, they require extensive resources until they are operational. CIIs, in contrast, are often not explicitly designed to be critical but become critical over time. Hence, they can grow in a flexible way and the initial costs are negligible.

Second, critical infrastructures and CIIs deteriorate in different ways. The strong dependence on hardware components in critical infrastructures makes them more prone to deterioration. For example, potholes must be filled, burst pipes must be replaced, and broken circuits must be rewired. CIIs, in contrast, do not deteriorate in similar ways. Over time, new bugs may be discovered but this is not deterioration of a CII. CIIs are deployed with shortcomings that need to be fixed over time. In other words, CIIs do not deteriorate, rather, they develop and improve.

Third, critical infrastructures and CIIs evolve differently over time. Critical infrastructures typically operate in relatively stable environments and their strong dependence on hardware components makes them more rigid and inflexible. Consequently, critical infrastructures evolve slowly. For instance, it takes years to increase the number of lanes on a highway in densely populated areas. CIIs, in contrast, operate in dynamic online environments and are constantly changing to meet novel demands. Accordingly, CIIs necessarily evolve rapidly to meet the changing requirements and growth targets.

Fourth, critical infrastructures are more predictable than CIIs. Without question, critical infrastructures are complex systems, which are difficult to grasp. Nevertheless, they are built for specific purposes with detailed design and development processes and they are rarely employed for secondary uses. For instance, the purpose of the power grid is to ensure a reliable electricity supply; trying to change this would not make much sense. In most cases, the consequences of critical infrastructures can be assessed and measured. Potential consequences of CIIs are often undeterminable. The dynamic environment of CIIs, their rapid evolution, and their intangible nature

make it hard to obtain an overview of all technical components, all subsidiary systems, all actors, and all affordances of CII. Accordingly, consequences of CII operation, especially, adverse ones, are often unforeseeable.

In a nutshell, critical infrastructures and CII create similar value for and have similar effects on society, but they differ in their design and operational characteristics. The term critical infrastructures can apply to many kinds of systems. The term CII, on the other hand, is specifically focused on specific kinds of information systems. Accordingly, critical infrastructures and CII should be treated as related, but different phenomena. The next section discusses the properties of CII in more detail.

11.2 Properties of Critical Information Infrastructures

CII are complex information systems that involve a wide range of actors and different technical components. It is therefore difficult to understand CII in their entirety, especially, since they manifest themselves in various forms and are generally not designed or perceived as CII from the outset. CII become increasingly important over time through their dissemination, ongoing evolution, and continued use in society. However, all CII share key characteristics that are presented below and summarized in Fig. 11.3.

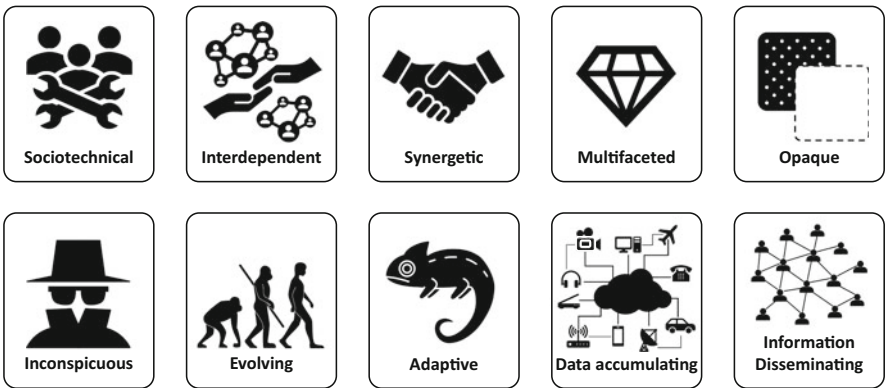


Fig. 11.3 Key characteristics of critical information infrastructures.

Sociotechnicality

As already described in section 11.1.2, CII are sociotechnical systems that comprise a variety of social and technical components, including technical structures, human resources, organizational processes, and laws and regulations. The work of setting

up a CII requires joint reflection on the components of its social and technical subsystems. Otherwise, important interdependencies will be neglected. This is, for instance, apparent in the case of flash crashes at stock exchanges. Flash crashes are often amplified by high-frequency trading. In such situations, automated trading algorithms often react to market movements in an irrational way, which triggers similar behavior among other automated traders. As a result, markets can crash within seconds or minutes. Once human traders have had a chance to catch up, their more rational actions often lead to the quick recovery of markets that flash-crashed.

Interdependency

The boundaries of a CII are often ambiguous and difficult to define because CII are composed of several social and technical components that are (in-)directly linked to each other. As a result, minor changes in different components of the CII can have unpredictable, and even catastrophic, consequences for parties that use and depend on the CII, due to the complex network created among the CII components. For example, health information infrastructures are often established and maintained on a national level and not by patients themselves. But, what happens when patients have treatment in different countries or when global pharma concerns become involved for research and development purposes? Should health information infrastructures be considered as one global CII, overlapping national CII, or distinct national CII? Within the CII context, answering such questions is not a straightforward matter—they require careful elaboration.

Synergies

CII are synergetic systems that create value beyond the sum of the values created by their individual components. For example, safety communication in vehicular ad hoc networks, commonly referred to as VANETs, becomes more and more useful as the number of participating cars increases. A single car could broadcast warning messages, but nobody would hear them. A small number of cars could hardly exchange warning messages, because they are unlikely to meet on the road. However, almost full coverage of cars would create an unparalleled information resource and communication infrastructure that would provide almost all drivers with real-time, geolocated safety information.

Multifaceted Systems

CII perform more than one task and are perceived differently by different actors. They are, therefore, multifaceted information systems that serve different objectives for different actors, without a central governance body. For example, a social networking service is used for a variety of purposes, such as individual communication, seeking personal information, self-presentation and promotion, data collection for personalization purposes, advertising, and generating revenue for the provider.

Opacity

The many components of CII and their complex relationships and interdependencies make CII opaque. Although it is relatively easy to identify some CII objectives, it is difficult to draw the complete picture. Moreover, it is not always

easy to understand how the different technical and social components interact. Towards the end of the second millennium, the notorious Y2K bug provoked a lot of attention. At the time, years were often specified by only two digits (e.g., 99 instead of 1999) in software code. Consequently, many people feared that the turn of the millennium would bring digital chaos because some computer systems might act as if it were the year 1900 instead of 2000. Worldwide, more than \$300 billion were spent on source code reviews and revisions. However, on New Year's Day, the global financial system did not collapse, no planes crashed, and no nuclear power plant spiraled out of control. To date, it is still undecided whether no catastrophes happened due to the extensive precautions or to the irrelevance of the Y2K bug.

Inconspicuous Systems

Since CIIs are generally not designed to be critical and, rather, become critical over time, they often operate without being noticed. Their importance might only become evident when they are disrupted or when negative consequences arise. For example, after the Cambridge Analytica scandal (Cadwalladr and Graham-Harrison 2018), many people were surprised to see how easy it was to use social media to manipulate public opinion. The nodes of the Mirai botnets, described in section 11.1.3, also had been latent on the Internet for quite some time before someone recognized and seized the opportunity and used them to create a network capable of creating massive DDoS attacks.

Evolving Systems

CIIs evolve over time for a variety of reasons. Developers add code to enhance offered services and add new features. Obsolete components are replaced by new technologies. New actors start participating in CIIs, using such structures for new purposes. New laws and regulations can require changes to business procedures or can change the purposes for which CIIs can be legally used. Web search started, for example, as a simple service that matched websites to queries specified in user interfaces. Over time, search infrastructures have evolved. Google started, for instance, to personalize search results based on users' past behavior on the Internet, similar users' behavior on the Internet, and the activity of users' friends in social networking services. Moreover, search results depend on the link structure between websites, so that web searching is constantly evolving through activity, providing new services offerings on the Internet.

Adaptive Systems

Thanks to the modularity and variety of their components, CIIs are, to a certain degree, adaptive. In the event of certain components failing, their functions can be taken over by other technical or social components. The challenge is to examine and understand redundancies in CIIs, to understand how a CII is best managed to handle unforeseen events, and to find effective ways of avoiding disruption and negative effects of CIIs. This is illustrated by the DAO (decentralized autonomous organization) hack. The DAO was a smart contract running on the Ethereum blockchain. In essence, the DAO represented an autonomous venture capital fund. Until the hack,

which happened one and half months after its launch, the DAO had already collected around \$150 million in Ether (the cryptocurrency in the Ethereum network). On June 17, 2016, a clever hacker exploited a combination of weaknesses in the DAO's source code to siphon off \$50 million in Ether. This showed that the DAO was not operating as intended, but remedial actions could not easily be taken because smart contracts are immutable. As a solution, the Ethereum network was hard forked¹ so that the investors whose Ether had been affected by the hack could be reimbursed. Hence, the failure of the DAO's smart contract was resolved by means of governance mechanisms and human intervention.

Data Accumulation

The main feature of CII is that they process information. Importantly, CII collect huge amounts of data over time, which requires not only sophisticated data storage and processing technologies but also careful development of the necessary information security measures and measures for preventing privacy violations. For example, in 2010, the American telecommunications company AT&T transmitted about 19 petabytes each day (AT&T 2010). By March 2018, this number increased by an order of magnitude to 197 petabytes each day (Gallagher and Moltke 2018). Growing sensor networks, especially the IoT, was a key driver for increasing the volume of collected data. The IoT was estimated to consist of 6.4 billion devices in 2016 and this number is projected to grow to 20.5 billion devices by 2020 (Meulen 2017).

Information Dissemination

As CII are essentially interconnected networks of social and technical components, they are very effective in disseminating information. Once CII have access to new information, the information can rapidly be shared with all other CII actors. After Stanley Milgram's 1969 experiments which studied the average length of social network paths in the USA, the term 'six degrees of separation' became popular. This term articulates that everyone on the planet is separated from every other by no more than six other people (Travers and Milgram 1977). Given the current high levels of interconnection, the media often report much lower separation levels: On Facebook, a separation of 3.74 degrees was calculated (Bhagat et al. 2016), indicating that information reaches people in the world much faster than ever before.

The characteristics outlined above shed more light on the nature of CII. In practice, CII will manifest in very different forms and for different purposes. Nevertheless, they all feature the characteristics presented above. However, it depends on the respective CII and its functions which characteristics are most pronounced and relevant.

¹ A hard fork constitutes a split of the community operating the validating nodes in a blockchain due to a protocol change that is not backward compatible. One subset of the community starts using the new protocol and the other keeps following the old protocol. The hard fork after the DAO hack split the Ethereum blockchain, for instance, into the Ethereum and Ethereum Classic blockchains.

11.3 Functions of Critical Information Infrastructures

CIIs serve various functions and exist in various forms. Some CIIs are critical due to detrimental effects that arise from their disruption (e.g., a control system in a nuclear power plant). Other CIIs are critical due to unintended consequences that result in detrimental effects (e.g., public opinion being manipulated through social networking services). Moreover, individual CIIs often perform multiple functions. In this section, we classify CIIs' functions into four categories (Fig. 11.4) determined by communication, governance, knowledge management, and information collection, as discussed below.

11.3.1 Communication

Communication infrastructures transfer information between humans and/or machines, and they perform three main functions. First, machine communication infrastructures are predominantly designed for communication between machines. An example of a machine communication infrastructure is the global navigation satellite system Galileo, which is a positioning system operated by the EU. Scientists started to develop this multibillion Euro infrastructure in the late 1990s, but it only started to deliver services in 2016 and is supposed to reach full operating capacity in 2019. The core of the Galileo infrastructure is made up of thirty satellites in the medium earth orbit. The satellites can be contacted by devices on earth to determine the contacting device's location. This is, for instance, useful for traffic and navigation applications. Galileo provides free and commercial services. A free positioning service is provided with a positioning accuracy of one meter. The commercial alternative, which is available at a fee, offers a positioning accuracy of one centimeter. Other Galileo services include a search and rescue feature which detects locations of distress beacons. A machine communication infrastructure, such as the information infrastructure Galileo offers, can be critical in that navigation satellite systems are used globally and provide services to private actors (as in road navigation), business actors (as in coordination of logistics), and state actors (as in coordination of military activities), thus exhibiting *critical breadth*. Disruption of Galileo could result in, e.g., economic losses due to the productivity of supply chains being impaired, or the weakening of countries' military power due to their reliance on Galileo for coordinating military activities, thus having effects of *critical magnitude*. Due to the dependence on satellites, which cannot easily be repaired or restored since they are in space, restoring a system like Galileo to full operating capacity can be time-intensive, thus having an effect of *critical duration*.

Second, private communication infrastructures support private communication between a limited group of persons. Such an infrastructure is, for example, created by the mobile application Signal. This free and open-source application runs on the

two major mobile operating systems Android and iOS. It is maintained by the organization Open Whisper Systems, which is funded through donations and grants. The application provides various communication features, such as text, voice, image, and one-to-one or group video messages. Communication with the application is kept private through public-key cryptography. Users generate a public and a private key locally on their device. Keys are used to verify the identity of communication participants. Every participant is identified by their public key. Furthermore, all messages are encrypted with recipients' public keys so that only intended message recipients are able to decrypt and read messages. Private communication infrastructures, such as the information infrastructure created through Signal, can be critical, as when, due to rising digitalization and globalization, people worldwide increasingly communicate via digital messengers instead of face-to-face, thus exhibiting *critical breadth*. Potential damages through disruption of communication pathways fit the criteria of *critical magnitude* because communication constitutes a cornerstone for functioning democratic societies. Moreover, people's communication practices are dependent on set habits and require communication participants to agree on compatible protocols (e.g., compatible messenger apps), so that it takes time for people to change communication pathways, showing an effect of *critical duration*.

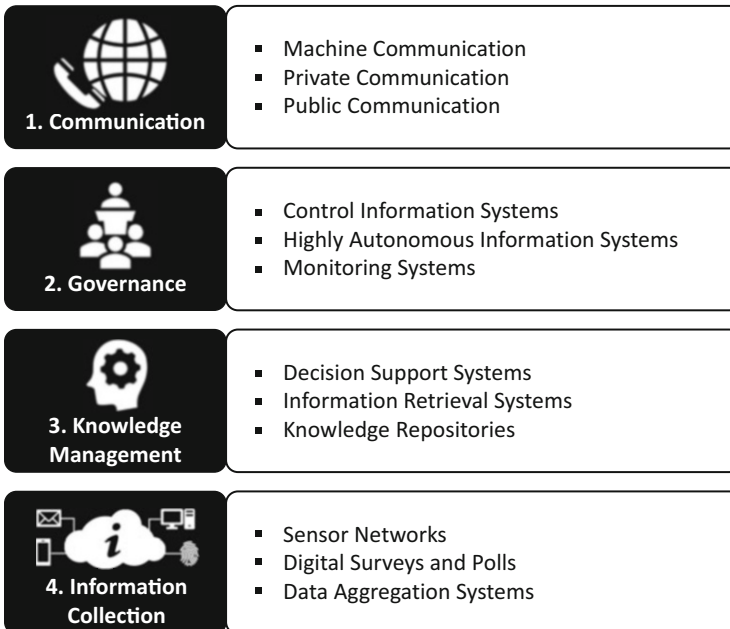


Fig. 11.4 The four categories of functions performed by critical information infrastructures.

Third, public communication infrastructures are used to share information with the general public. The Emergency Alert System (EAS) in the USA provides an example of a public communication infrastructure. The EAS is jointly operated by the US Federal Emergency Agency (FEMA), the US Federal Communications Commission (FCC), and the US National Oceanic and Atmospheric Administration (NOAA). The EAS's main objective is to allow the President of the USA to communicate to all their citizens within 10 minutes. Moreover, authorized organizations can use the EAS to publicly broadcast alerts about emergencies, such as tornados, flash floods, or severe thunderstorms. The EAS communicates information via terrestrial and satellite connections to radio and television receivers and via cellular networks to cellphones. Public communication infrastructures, such as the information infrastructure offered by the EAS, can be critical in that the EAS is designed to communicate information to all US citizens, thus exhibiting *critical breadth*. Disruption of the EAS would delay due warning of US citizens about imminent threats, thereby increasing the extent of destroyed property and the potential death toll, therefore having effects of *critical magnitude*. Such adverse consequences are difficult, if not impossible, to remedy, thus showing an effect of *critical duration*.

11.3.2 Governance

Governance infrastructures are information systems that control and monitor other infrastructures. Governance infrastructures perform three main functions. First, control information systems ensure that infrastructures stay within defined control parameters. Systems that instantiate the supervisory control and data acquisition (SCADA) architecture provide an example of control information systems. SCADA systems are used to control critical infrastructures such as water supply, electric power grids, or oil pipelines. SCADA systems implement a layered architecture to control diverse devices. The bottom layer consists of sensors and actuators in controlled devices in which the sensors are used to collect information about the operation of controlled devices, and the actuators are used to execute control commands that influence the operation of controlled infrastructures. The second layer provides the SCADA interface for the controlled devices and provides the functions required to monitor and configure the controlled devices. The third layer, aggregates information about multiple controlled devices for local operators. The fourth and the fifth layers provide higher-level aggregations of information about the controlled devices to facilitate management of all controlled devices and long-term planning. Control information systems, such as SCADA systems, can be critical as they control other systems (e.g., power grids) that provide important services to many people, companies, and governmental actors, thus exhibiting *critical breadth*.

Failure of a SCADA system impedes the operating capability of the controlled systems, which can result in adverse consequences of *critical magnitude* (e.g., if widely distributed traffic lights stop working during blackouts). SCADA systems are usually employed to simplify and automate the operation of systems that are too complex or large to be directly controlled; hence, finding errors in SCADA systems and restoring normal operating capability is time-consuming, thus showing an effect of *critical duration*.

Second, highly autonomous information systems perform tasks within an infrastructure. High frequency trading provides an example of highly autonomous information systems. The digitalization of stock exchanges in the 1980s allowed investors to trade securities purely electronically. Accordingly, some traders started to implement algorithms that trade autonomously according to predefined rules and strategies. Systems for high frequency trading require market data as input. Such data is usually supplied either in the form of current prices and open orders for securities, or as a list of all orders placed by market participants. Systems for high frequency trading can execute two actions: They can place orders to sell or buy securities, and they can cancel orders. The basic strategy is to leverage the advantage that computers can act on new data faster than human traders. Systems for high frequency trading use that head start to place orders before human traders can do so. In addition, they do not hold securities for long and so have a quick turnover. High frequency trading does not create high profit on single trade deals; instead, a large number of trade deals with small margins can be conducted in a short time. Time, or rather latency, is a crucial factor in high frequency trading. Successful high frequency trading requires rapid access to high quality market data, short processing times to analyze new data and make trading decisions, and short delays when orders are placed at exchanges. Highly-autonomous information systems, such as systems for high frequency trading, can be critical in that stock exchanges establish financial ties between a wide range of industries, thus exhibiting critical breadth. Also, poor foresight during the design or manipulation of systems for high frequency trading can lead to irrational stock price developments, which can result in adverse consequences for entire industries, which demonstrate effects of *critical magnitude*. Due to the enormous speed of high frequency trading, adverse consequences manifest before human actors have a chance to detect deviations from the expected operation, to identify the causes, and to rectify the fallout, thus showing an effect of *critical duration*.

Third, monitoring systems watch control parameters and raise alarm in cases of violation. An intrusion detection system is an example of a monitoring system. Such systems operate in computer networks to identify malicious traffic that, once identified, alerts human operators or other systems so that remedial steps can be taken. The dominant approaches to detecting malicious traffic are signature-based and anomaly-based detection. Signature-based detection maintains a database of known attacks and raises alarm as soon as new packets with signatures that match known attacks are identified. Anomaly-based detection establishes a baseline of benign network traffic and raises alarms once unusual traffic patterns are identified. The disadvantage of signature-based detection is that it can only detect known attacks and that when new attacks occur, updating the database takes time.

Anomaly-based detection, in contrast, can detect unknown attacks but it is prone to producing false positives due to benign traffic patterns that only rarely occur. Hence, a combination of signature-based and anomaly-based detection is often employed. Monitoring systems, such as intrusion detection systems, can be critical in that the purpose of monitoring systems is to alert human actors to deviations from normal operations; hence, their disruption can prevent human actors from timeously recognizing problematic states of operation, thus having an effect of *critical duration*. If problems remain unnoticed, their adverse consequences cannot be mitigated and could even aggravate the difficulty. Hence, disruption of monitoring systems also increases the *breadth* and *proportion* of resulting damages because remedial steps cannot be taken early on.

11.3.3 Knowledge Management

Knowledge management infrastructures preserve information for future use. Such infrastructures perform three main functions. First, decision support systems provide users with information to improve decision-making. For example, clinical decision support systems are designed to support clinicians in clinical decision-making by combining the advantage that computers can rapidly process vast amounts of information with the intelligence of users. Based on current patient data, clinical decision support systems propose alternative diagnosis and treatment strategies based on state-of-the-art scientific knowledge. Clinicians screen the proposed alternatives and choose the best alternative based on their expertise and their patient's preferences. Clinical decision support systems must be seamlessly integrated with clinical workflows so that they can raise alarm or make suggestions whenever required and directly at the point of care. In pharmacies, clinical decision support systems are, for instance, useful in detecting undesirable drug interactions in prescribed regimens. The three main components of clinical decision support systems are the knowledge base, the inference engine, and the user interface. The knowledge base stores state-of-the-art medical knowledge encoded in rules (e.g., if taking pill A, do not take pill B). It is usually centrally maintained and updated by the system provider to relieve users of the burden of keeping track of all new developments themselves. The inference engine connects the rules in the knowledge base to the patient's current electronic health records, identifies warnings, and makes suggestions accordingly. These are displayed to clinicians via a user interface. Usually, clinical decision support systems do not use stand-alone user interfaces. Instead, they are integrated with the systems that clinicians use during their normal workflow. Decision support systems, such as clinical decision support systems, can be critical in that they influence the quality of care of all medical practitioners that use them around the world, thus exhibiting *critical breadth*. Errors in clinical decision support systems can result in poor medical decisions which directly impact patients' state of health, thus having effects of *critical magnitude*. Poor clinical decisions (e.g., treatment with the wrong drug) can have bad consequences (e.g., worsened state

of health or death due to drug overdose) which are hard or impossible to remedy, thus showing an effect of *critical duration*.

Second, information retrieval systems facilitate the discovery and retrieval of information. The web search engine of DuckDuckGo, Inc, which was founded in 2008, provides an example of an information retrieval system. The company markets its web search engine based on privacy by giving assurance that it does not store personal information, does not use behaviorally targeted advertisements, and does not track users. The web search engine provides search results based on entered search terms and disregards information, such as IP address, identifiable cookies, or user agents. Consequently, all users are given the same search results for the same query (Holwerda 2011). This is a hybrid web search engine that builds on over 400 different sources, including its own search index, knowledge repositories (e.g., Wikipedia), and APIs of other search engines (e.g., Bing, Yahoo!, or Yandex). DuckDuckGo's web search engine implements a search strategy different to other web search engines. Instead of trying to provide users with the search results that best fit their data, this engine delivers the search results that best fit their search query. Information retrieval systems, such as web search engines, can be critical in that it is intractable for human actors to find the desired information on the Internet manually, so that almost all Internet users rely on web search engines to navigate the world wide web. This, again, exhibits *critical breadth*. Accordingly, web search engines influence what information is widely disseminated so that errors or malicious implementations can manipulate the public discourse, thereby having effects of *critical magnitude*). Such subliminal manipulations are hard to detect and rectify, so that there is also an effect of *critical duration*.

Third, knowledge repositories retain data, information, or knowledge. The online encyclopedia, Wikipedia, is an example of a knowledge repository. Wikipedia, founded on January 15, 2001 by Jimmy Wales and Larry Singer, is a community-driven project that aims to develop an online encyclopedia. Initially, it was an English-language encyclopedia, but it was quickly extended to use additional languages. By 2019, Wikipedia is available in more than 300 different languages and hosts over 40 million articles. With more than 5.5 million articles, the English-language version is the largest Wikipedia. Initially, everyone could arbitrarily add and edit articles in Wikipedia. Changes could be rolled back through a version history. Over time, a more elaborate governance system emerged due to vandalism and spam content. Wikipedia edits are now publicly mapped to IP addresses or user accounts to deter malicious actors and to be able to trace undesirable actions. Wikipedia editors gain status in the Wikipedia community over time, and those who make good contributions are elected to take over roles with more rights and responsibilities. In some language versions, edits by less reputable editors have to be reviewed by editors with higher status before they are published. Although Wikipedia has millions of different editors, almost half of the edits are made by only 1% of all editors. Most of these edits are corrections in writing and formatting. The actual content is produced by a larger group of editors. Wikipedia was the first successful collaborative online encyclopedia and it has now evolved into a knowledge repository that is among the most popular global websites. Knowledge

repositories such as Wikipedia can be critical. Since Wikipedia is among the most popular global websites, it affects many people, thus exhibiting *critical breadth*. Knowledge repositories determine what users accept as true; hence, they impact users' worldview, thereby gaining effects of *critical magnitude*. Adverse consequences from interference with people's worldviews (e.g., vaccine denial) often linger undetected for a considerable time and are hard to remedy, thus showing an effect of *critical duration*.

11.3.4 Information Collection

Information collection infrastructures harvest information for further processing by performing three main functions. First, sensor networks collect information on their environment. Ambient air quality monitoring systems collect, for example, long-term data on air quality. This data is collected for various purposes, such as to assess pollution levels, to inform the general public about air quality, to support the implementation of air quality standards and regulations, to assess the effectiveness of air quality policies, to detect trends in air quality levels, to develop and evaluate air quality models, and to make data available for research projects. Measurement stations are usually positioned in places where pollution levels are likely to be high. Such places could be industrial plants or city roads with high traffic. In addition, measuring stations are positioned in places where pollution levels should not be high, such as hospitals or schools. The US Environmental Protection Agency (EPA), for instance, operates various air quality monitoring systems, to measure airborne toxics, to track lead pollution, to collect meteorological measurements, and to measure nitrogen dioxide or ozone levels. Sensor networks, such as ambient air quality monitoring systems, can be critical in that digital data collection has resulted in a radical increase of available data, thus becoming a key resource for decision-making and so exhibiting *critical breadth*. This includes decisions that shape public policy thus having effects of *critical magnitude*. Moreover, it is difficult to retroactively determine the correct value of a data point at a specific time if its measurement was biased or manipulated, thus showing an effect of critical duration.

Second, digital surveys and polls collect information about the opinions of a group of people. Electronic voting provides an example of digital polling. Electronic voting machines aim to make voting cheaper and more efficient by reducing the need for paper-based work. Machines for electronic voting are designed in different ways. Some machines use punch cards so that votes are still marked by hand but can be counted electronically afterwards. Other voting machines, such as direct-recording electronic (DRE) machines, are stand-alone solutions. DRE machines display possible candidates or parties on a screen and let voters cast their vote. Afterwards, the machines store each vote in local memory and print a receipt for the voter. Once the election is over, the voting results can be printed and further processed. Public network DRE voting systems connect multiple DRE machines in different precincts via a public network, such as the Internet, and use a central server to directly count

the votes entered on all connected DRE machines. Other solutions are web-based, so that voters can cast their votes directly from their browser. In Estonia, for instance, every citizen has a national ID card, which they can use to verify their identity online. Estonians can use their national ID card to register online in the Estonian electronic voting system and to cast their vote after successful identification. Digital surveys and polls, such as electronic voting, can be critical, as when electronic voting employed for elections determines, for instance, who will govern, thus having an effect of *critical magnitude*, across a nation, thus exhibiting *critical breadth*. As governments remain in power for a term of multiple years, there is an effect of *critical duration*.

Third, data aggregation systems generate information from data streams. Google Flu Trends, a website operated by Google from 2008 to 2015, provides an example of a data aggregation system. The website analyzed search queries made with Google's web search engine to predict influenza outbreaks in the USA. The website was trained with data from 2003 to 2007. The data set was analyzed to identify search queries that correlated highly with influenza prevalence as reported by the US Centers for Disease Control and Prevention (CDC). Initially, Google Flu Trends performed quite well and produced accurate forecasts a couple of days earlier than alternative prediction models. However, over time the performance of Google Flu Trends declined, so that the project was abandoned in 2015. The failure of Google Flu Trends is ascribed to an overfitting of the search query data to the influenza prevalence data. The correlation analyses also identified other seasonal terms that correlated by random chance with influenza prevalence (e.g., high school basketball). In addition, Google Flu Trends did not account for changes in users' search behavior, and Google skewed search behavior themselves through features of their web search engine (e.g., suggestions for related search terms). Data aggregation systems, such as Google Flu Trends, can be critical in that, in our interconnected world, output of data aggregation systems is often used as input for other systems whose correct functioning depends on the reliability of the provided input, thus exhibiting *critical breadth*. Due to unbalanced, erroneous, or missing input data, data aggregation systems can exhibit systematic biases (e.g., racial bias, gender bias) and intensify societal challenges, thus showing effects of *critical magnitude* in ways that are not easily traceable and corrected, which testify to *critical duration* effects. The case of Google Flu Trends shows that CIIs can serve as new data sources to derive novel insights or improve on extant solutions. However, data aggregation must be performed cautiously, and the quality and correctness of inferred inferences must continuously be ensured.

11.4 Operation of Critical Information Infrastructures

A salient characteristic that sets CIIs apart from information systems in general is that CIIs are critical. An example of an information system that is not critical could be an installation of a customer relationship management (CRM) system. Disrupting

a CRM system can result in reduced productivity, lost reputation, and financial losses, to the extent that recovery can require a considerable amount of time; however, the consequences do not achieve critical breadth as they impact only a single or a few companies. The fact that CIIs can produce consequences of critical magnitude, critical breadth, and critical duration creates some challenges with which other information systems are not necessarily confronted. In the following section, we discuss eleven challenges that have to be addressed carefully during CII operation (Fig. 11.5) and that outline starting points for dealing with the difficulties.

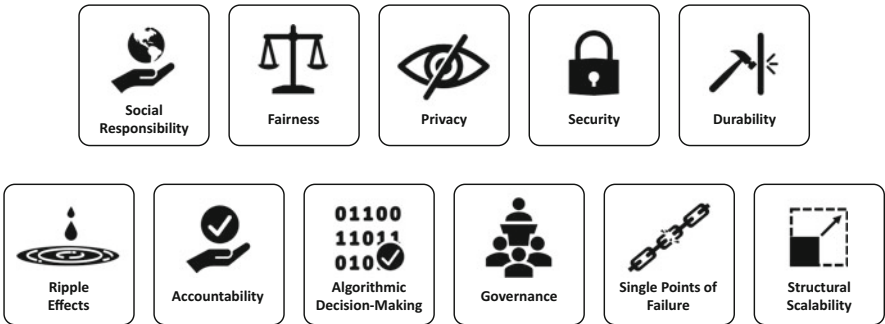


Fig. 11.5 Eleven challenges for that have to be addressed carefully during CII operation.

Social Responsibility

By definition, CIIs fulfill roles that are important to society. Accordingly, they cannot be managed like conventional, commercial information systems. CIIs will not fulfill their functions if their management merely seeks to create economic value and fails to pay attention to CIIs’ important roles in society. Ideally, CII operation would be guided by social responsibility. Social responsibility “encompasses the economic, legal, ethical, and discretionary expectations that society has of organizations at a given point in time”(Carroll 1979).

In general, organizations can follow five strategies to respond to social pressure (Oliver 1991): manipulation, defiance, avoidance, compromise, and acquiescence. Manipulation is the most uncooperative strategy as it refers to proactive engagement to change social pressures in a way that aligns with the organization’s goals. Defiance is a strategy in which organizations ignore social pressures and proceed as they see fit. Organizations wanting to avoid social repercussions employ avoidance tactics in attempting to conceal their non-compliance with social pressures. The compromise strategy trades off organizational objectives against external social pressures, attempting to some extent to satisfy both. Acquiescence is the most compliant strategy where organizations align their processes with social values and expectations. Due to their importance in society, CII operators are most likely to succeed when they adopt strategies geared toward compliance, that is, if they follow acquiescence or at least compromise strategies. Less cooperative strategies,

which are viable options in normal organizations, are not useful to CII operators. Hence, CII operators often have to change an organizational culture once the information infrastructures they operate become critical.

Fairness

Besides operating in a socially responsible way, in order to fulfill their role in society, CIIs must also be perceived as fair. CIIs should impact all people equally and should not violate human rights. Fairness is closely tied to justice considerations, which generally means that CII operators must cater for three modes of justice to be perceived as fair (Cropanzano et al. 2001), namely distributive justice, procedural justice, and interactional justice. Distributive justice refers to the fair allocation of outcomes a CII produces. CIIs must be designed in such a way that they do not unduly privilege or disadvantage any actors in a society. Important norms that can be chosen to achieve distributive justice are equity, equality, power, need, and responsibility. Equity ensures that all affected parties carry an equal share of the costs and benefits of CII operation. Equality ensures that rewards from CII operation are obtained in relation to the input a party has contributed. Power distributes more outputs to those parties with higher power in the society. Need distributes more outputs to those parties that are in greatest need of them. Responsibility refers to sharing of outputs by those who have more with those who have less. Procedural justice refers to the establishment of fair processes for decision-making. In the legal domain, procedural justice is, for instance, instantiated through due process. In contrast to distributive justice, procedural justice focuses not on the distribution of the outcomes but on the processes that were used to distribute outcomes. Interactional justice focuses on dignity and respect in the relationship between decision makers and the people affected by the decisions. Distributive, procedural, and interactional justice are interdependent concepts (Cropanzano et al. 2001). Shortcomings in any of the different types of justice can be remedied through enhancement in another type. CII operators should not simply strive to make CIIs as fair as possible; rather, the different types of justice must be considered and instituted in a way that will make CIIs fair enough.

Privacy

The concept of privacy is constantly evolving with social change. While in ancient Greece revealing the naked body was acceptable as it was perceived as a sign of being civilized, rising wealth and less dependence on servants increased demands for hiding one's private parts (Solove 2002). Within the CII domain, a similar development driven by technological change is evident. In the late 19th century, increasing prevalence of newspapers led to a rights-based conceptualization of privacy (Warren and Brandeis 1890). In the 1960s, computerizing governmental databases led to another change in the conceptualization of privacy, regarding individuals' control over communication of personal information (Westin 1968). Then, with the commercialization of information infrastructures, market-based conceptualizations of privacy, which treated information as a commodity, emerged (Laudon 1996).

Today, we live in a globally-connected world where information collection has become virtually ubiquitous. Hence, across borders, rights-based conceptualizations

are hard to enforce; also, they adapt too slowly to the rapid technological innovations we experience (Oetzel and Spiekermann 2014). Control-based conceptualizations of privacy do not account for the sheer complexity of the current information environment and the large amount of user interactions that accompany it (Landau 2015). Market-based approaches do not foster privacy because they promote opportunistic behavior of information handlers. The latter can leverage the enormous potential of hidden information and hidden action (Clarke 1999).

A more promising approach than a rights-based or control-based one, considers privacy in today's highly interconnected information infrastructures within a contextual integrity framework, which focuses on the appropriateness of information flows (Nissenbaum 2010). CII users judge appropriateness of information flows according to information norms that have five characteristics (Martin and Nissenbaum 2016), namely subject (i.e., to whom the information pertains), sender (i.e., who sends the information), recipient (i.e., who receives the information), information types (e.g., phone number, purchase, medical condition), and transmission principle (e.g., in confidence, with consent). Each of the five characteristics can take on a wide range of values, which results in a vast five-dimensional space of possible contexts. CII operators must master this complexity to be able to respect privacy during CII operation.

Security

Information security requires that the principles of confidentiality, integrity, and availability (CIA triad) are upheld (Dehling and Sunyaev 2014). The CIA triad that has become a generally accepted framework for information security. To secure confidentiality, CII must ensure that only authorized users have access to information. To maintain integrity, stored information must be protected from unauthorized modifications or deletions and from random and unwanted changes by authorized users. Regarding availability, CII must be available and fully functional whenever users desire to use them. Besides the CIA triad, CII operators have to be aware of potential attackers and their motives (Sunyaev et al. 2008). Possible sources of attacks are, for example, hackers, script-kiddies, malicious insiders, members of organized crime, common malware, (cyber)terrorists, or the media. The reasons for such attacks vary. Some attackers may be motivated by financial incentives and others by boredom or curiosity. In any way, security is an ongoing challenge for CII operators that has to be addressed diligently (Kannengießer et al. 2019).

Durability

Once they have become critical, CII often persist. Accordingly, they need to be designed in a way that makes them durable. CII should leverage modularity so that broken components can be replaced quickly. To keep operating expenses low, CII should build on components that require limited maintenance. In a perfect scenario, CII would not require significant adjustments over time. However, the dynamic nature of information technology makes durability an exceptional challenge. Potential future developments should be considered as early as possible in CII design, but CII operators also have to focus on change management to adapt CII to unforeseen environmental circumstances. Common maintenance tasks

include, for instance, applying security patches and adapting to evolving legislation. That information technology is subject to rapid evolution, adds to the challenging aspect of durability.

Ripple Effects

CIIs are usually embedded in complex networks. Hence, they cannot be treated like stand-alone systems. Instead, their effects on interconnected systems must be taken into account. Changes to or disruptions of a CII can create effects that spread to other systems and (critical) information infrastructures.

CIIIs can exhibit three types of interdependency (Rinaldi et al. 2001), namely upstream dependency, internal dependency, or downstream dependency. Upstream dependency refers to a situation in which a CII depends on outputs produced by another infrastructure. CIIIs are, for instance, often dependent on Internet access. Internal dependency refers to dependencies between the components of a CII. A social networking service must, for instance, deal with interdependency between the number of current users and available hardware resources. Downstream dependency captures other systems or persons that are dependent on the operation of a CII. Without the Global Positioning System (GPS), many traffic and navigation services would, for instance, cease to operate.

Dependencies can surface in four different forms (Rinaldi et al. 2001), identified as physical, geographical, cyber, or logical dependencies. Physical dependencies refer to functional or structural linkages between systems. A CII can, for example, be dependent on electricity provided by the power grid. Geographical dependencies refer to events in the local environment that can impact CII operation. Hardware required for CII operation can, for instance, be affected by natural disasters, such as an earthquake or a flood. Cyber dependencies capture dependencies on electronic or informational links. An information retrieval system depends, for instance, on data about all the information that could potentially be retrieved. Logical dependencies capture all dependencies that are neither physical, geographical, nor cyber dependencies and can usually be attributed to human factors. CIIIs are, for example, often dependent on the current legal system. Due to their complex nature, CIIIs' ripple effects are often hard to foresee. Nevertheless, CII operators cannot pay attention to their own CII only; they have to consider the environment in which their CII operates.

Accountability

Accountability refers to being responsible for one's actions. Within the context of CIIIs, accountability constitutes a challenge because it is often hard to identify the person or organization who is responsible for certain functions.

Four barriers impede accountability in a computerized society (Nissenbaum 1996), namely the problem of too many hands, bugs, the computer as a scapegoat, and software ownership without liability. The problem of too many hands refers to the fact that CIIIs are operated and designed by diverse actors, are interdependent with other infrastructures or systems, and reuse components (e.g., software libraries). This makes it hard to identify the parties who were at fault in situations where things go wrong. Bugs are inevitable in software components, especially in infrastructures

as complex as CII. CII developers or operators can technically be held accountable for incompetent actions. However, requiring software or hardware developers to eradicate all bugs would make development prohibitively expensive. This is a challenge for CII because it is difficult to decide whether adverse consequences should be attributed to the actions of individual parties or to general risks of employing computer technology. The computer as scapegoat captures situations where blame is attributed to computers instead of to the humans operating them. In the predigital past, many functions that are currently fulfilled by CII where tasks executed by human actors. Today, CII determine, for example, which news articles we will find, or they decide autonomously which stocks to sell or buy. Since CII perform functions previously performed by human actors, people tend intuitively to blame software for adverse consequences. This is, however, problematic because software cannot remedy wrongs. Accountability requires that blame be assigned to (legal) persons. Software ownership without liability captures the circumstance that many software vendors maintain ownership of the software they produce but do not take responsibility for consequences that result from using the software. With open source components this is even more problematic since either nobody owns the code, or everyone does. Especially with respect to software components in CII, it is hard to assign blame when things go wrong.

Algorithmic Decision-Making

Inevitably, CII comprise human and machine actors. However, with the rise of big data and immense progress in machine learning, CII are increasingly replacing human decision-making with algorithmic decision-making. This is an important development to increase performance, scalability, and fairness of CII. However, algorithmic decision-making is not necessarily unbiased or value free. A prominent example of a decision-support system that was racially biased is, for instance, the tool Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) that was used in the USA to assess recidivism risks of defendants in pretrial hearings (Caplan et al. 2018). The problem with the COMPAS system was that it produced more false negatives for white than for black people and more false positives for black than for white people; that is, COMPAS mistakenly judged white people who were recidivist to not be recidivist and black people who were not recidivist to be recidivist.

Different factors can cause bias in algorithmic decision-making (Caplan et al. 2018), such as embedded values, opacity, repurposing of data and algorithms, lack of auditing standards, power and control, and trust and expertise. Embedded values refers to values that programmers consciously or unconsciously embed in their code. Moreover, algorithms are written at a certain point in time and values that were current at the time of writing can become outdated over time. Opacity can impede traceability of algorithmic decision-making for three reasons. First, algorithms might not be disclosed for reasons of national security or to protect trade secrets. Second, auditors of algorithms might lack the necessary qualification to comprehend algorithms (e.g., lay users). Third, algorithms might become too complex so that humans cannot comprehend how the algorithms arrive at their decisions (e.g., in deep-

learning clinical decision support systems). Repurposing data or algorithms can create problems because data or algorithms might not exactly fit the decision-making context of the new algorithm. Nevertheless, this is done because to curate training data and develop algorithms is costly. Auditing standards are hard to develop because algorithms are used in diverse contexts, which makes it difficult to find global consensus on which social values algorithms should not violate. The factor of power and control refers to challenges that arise from the institutionalization of algorithms. Over time, people become accustomed to algorithms. Thus, algorithms can be used to intentionally or unintentionally reshape social values. The Google search algorithm, for instance, reshapes how web content providers frame their content so that they are better listed on the Google index. The issue of trust and expertise refers to the challenge associated with humans intuitively trusting the output of algorithms because they value the expertise of their creators and perceive algorithms as less likely to fall for human fallacies (e.g., humans are often prone to trust other humans). However, algorithms are not free from bias, and therefore they need to be scrutinized more than is currently being done.

Governance

Governance refers to the design and practice of actions and processes that produce stable practices in organizations. Governance bodies exist on all levels of society (e.g., in supra-national bodies, national governments, regional governments, municipal governments, corporate management, or club management). Within the context of CIIs, a pertinent form of governance is information technology governance. Information technology governance is “the system to direct and control use of IT [...] from a business perspective, not an IT perspective” (Juiz and Toomey 2015).

Information technology governance is a continuous cycle encompassing three tasks (Juiz and Toomey 2015). First, the use of information technology in the organization and associated business strategy is evaluated. Second, plans for changes to information technology policy are assessed. Third, information technology policy is implemented and monitored. Good information technology governance should adhere to six principles (Juiz and Toomey 2015), which are responsibility (i.e., assigning responsibilities for decisions on information technology use and supply), strategy (i.e., aligning information technology supply and use with organizational goals), acquisition (i.e., investing in new and required information technology), performance (i.e., matching information technology capabilities with business needs), conformance (i.e., ensuring that information technology does not contradict formal rules and regulations), and human behavior (i.e., respecting human behavior when deploying and using information technology).

Information technology governance is a subset of corporate governance, which is concerned with the governance of corporations. Governance of information infrastructures becomes challenging once information infrastructures turn into CIIs, as CIIs must conform with social values and cannot be governed by corporate strategy only. The transformation of an information infrastructure into a CII will bring in new actors with new interests that must be accounted for in the CII governance. Moreover, CIIs are usually complex systems with many (inter-)

dependencies which makes it hard to devise, implement, and enforce effective governance strategies.

Single Points of Failure

To a certain degree, CII exhibit redundancies. A failing database can, for instance, easily be replaced by a redundant database. From a technical perspective, redundancy can be achieved on the level of building blocks of CII components (e.g., redundant power supplies in servers), on the level of individual components (e.g., replicated databases), or on the level of the whole CII (e.g., the GPS operated by the USA and the global navigation satellite system Galileo operated by the EU). CII operators must leverage redundancies to avoid single points of failure in a CII, because the disruption of a single point of failure would stop operation of the whole CII. This is a manageable task for simple components of a CII but becomes increasingly difficult with increasing complexity and scope of CII components. For example, it is easy to have redundant databases but it is much more complex to have a redundant positioning system. Establishing redundancy is also easier for technical components than for human components of a CII. To illustrate, the ‘bus factor’ is a measure for assessing how well a CII is protected from a human single point of failure. In principle, the bus factor captures how many people who can perform a particular task relevant to the operation of a CII, can be ‘hit by a bus’ in one go, or analogously, be incapacitated by some other unexpected event, before the CII is disrupted. Reasons that reduce the bus factor in software development are, for instance, obfuscated source code or undocumented complex code.

To avoid such single menacing factors, CII operation requires a work system that stresses flexible skill sets, reduces complexity, and provides up-to-date comprehensible and comprehensive documentation. The means to reduce the risks created by single points of failure are readily available. The main challenge for CII operators lies in identifying single points of possible failure. Especially when considering CII within their network of interdependent infrastructures and systems, it becomes difficult to make sure that all single points of failure have been identified and properly addressed.

Structural Scalability

In general, scalability refers to a system’s capacity to handle changing amounts of work without severe deterioration in performance or waste of resources. Scalability is an important objective for CII, because increased criticality is often accompanied by a workload increase, which coincides with an increase in the CII’s breadth.

There are four types of scalability (Bondi 2000), which are load scalability, space scalability, space-time scalability, and structural scalability. Load scalability describes the capacity of a system to avoid unnecessary delays and resource consumption under light, moderate, and heavy loads while using resources in an efficient way. Space scalability captures the capacity of a system to process increasing amounts of work without unmanageable increases in memory requirements. Space-time scalability is the capacity of a system to handle increases in work volumes by orders of magnitude without severe performance losses (e.g., use of search indices which are sublinear, instead of linear lists for information

retrieval). Structural scalability demands that adjustments required to handle increasing work volumes are not prevented by implementation or architecture specifications (e.g., the too small number of possible Internet Protocol version 4 (IPv4) addresses).

Although all scalability types are important for CII, structural scalability is the most compelling one within the CII context. CII need to provide consistent services to their users and other CII or systems might be dependent on the protocols and data structures implemented in a particular CII. Hence, changes to a CII's implementation or architecture can result in errors at interconnected CII and other systems. Therefore, CII must be structurally scalable without requiring changes in implementation or architecture specifications.

Overall, the challenges that CII are confronted with stress the sociotechnical nature of CII. Effective operation of CII requires that operators master not only technical challenges, but also ethical, legal, and social challenges. Accordingly, effective CII management constitutes an interdisciplinary effort where close collaboration is required to avoid decisions that improve CII with respect to one feature, but worsen others.

Summary

Information technology has developed impressively since the 1960s. Mainframes have been replaced by personal computers and mobile devices, the Internet has rapidly interconnected almost any device on a global scale and, nowadays, information technology has penetrated nearly all levels of economies, societies, and daily life. Some information systems have even become critical, that is, there will be adverse consequences of critical magnitude, critical breadth, and critical duration if they are disrupted or malfunction. Such information systems are commonly referred to as critical information infrastructures (CII). A CII can be defined as an information system that, if it is disrupted or has unintended consequences, can have detrimental effects on vital societal functions or the health, safety, security, or economic and social well-being of people. The emergence of CII has been propelled by the fifth technological transformation, which was characterized by the increasing dissemination of telecommunication and computer technologies from the 1970s onwards. CII, and information systems in general, are inherently sociotechnical so that a sociotechnical perspective has to be considered when people deal with CII. Key components that have to be considered in managing or changing CII are the technical subsystems with their physical systems and tasks and the social subsystems with their structures and people.

CII are considered to be related to critical infrastructures. However, unlike critical infrastructures, CII focus on information processing and flows within infrastructures. Critical infrastructures, such as communication networks, are of marginal importance in the area of CII. Instead, they create the necessary environment for the creation and functioning of CII. In a nutshell, critical infrastructures

and CII have similar values and effects on society, but they differ in their design and operational characteristics. The term critical infrastructure can refer to all kinds of systems. The term CII, on the other hand, is more focused and concentrates on information systems. Accordingly, critical infrastructures and CII should be treated as related but different phenomena.

CII are complex information systems involving a wide range of actors and different technical elements. It is therefore difficult to understand CII as a whole, especially since they manifest themselves in various forms and are generally not perceived as, or intended to be, CII from the outset. CII become increasingly important over time through their dissemination and continued use in society. They become more and more critical as the number of users and affordances they offer, grows. However, all CII share ten key characteristics: sociotechnicality, interdependency, synergies, multifaceted systems, opacity, inconspicuous systems, evolving systems, adaptive systems, data accumulation, and information dissemination. It depends, however, on the particular CII and its functions which characteristics are most pronounced and relevant in a given case.

CII serve various functions and exist in various forms. Moreover, individual CII often perform multiple functions, of which the main categories of functions they offer are communication, governance, knowledge management, and information collection. Communication infrastructures transfer information between humans and/or machines and can be classified into the three subtypes machine communication, private communication, and public communication. Governance infrastructures are information systems that control and monitor other infrastructures, and these can be classified into the three subtypes control information systems, highly autonomous information systems, and monitoring systems. Knowledge management infrastructures preserve information for future uses and can be classified into the three subtypes decision support systems, information retrieval systems, and knowledge repositories. Information collection infrastructures harvest information for further processing and can be split into the three subtypes sensor networks, digital surveys and polls, and data aggregation systems.

A salient characteristic that sets CII apart from information systems in general is that CII are critical for the running of established societal systems. This sole fact creates eleven challenges other information systems may not necessarily be confronted with: First, CII operators must pursue social responsibility. Second, CII must account for fairness. Third, CII require strong privacy protection mechanisms. Fourth, security is an important requirement for successful CII operation. Fifth, CII must be designed to be durable. Sixth, CII must account for ripple effects. Seventh, CII require accountability mechanisms to identify responsible parties in case of CII disruption or the manifestation of unintended consequences. Eighth, careful attention must be paid to algorithmic decision-making. Ninth, the complexity of CII makes governance more challenging in contrast to other information systems. Tenth, single points of failure are the Achilles' heel of CII and should be avoided. Eleventh, CII require careful design decisions to allow for structural scalability.

To effectively operate CIIs, operators are required who can master not only technical challenges but also ethical, legal, and social challenges. Accordingly, effective CII management constitutes an interdisciplinary effort where close collaboration between different actors is required to avoid decisions that improve CIIs in one respect but worsen them in others.

Questions

1. What were the key drivers of the five technological transformations identified in history, and how did they impact social life?
2. What are sociotechnical systems, and what is the difference between a sociotechnical and a sociomaterial perspective?
3. What are critical information infrastructures?
4. What are the differences between critical infrastructures and critical information infrastructures?
5. How does the importance and relevance of critical information infrastructures' main properties differ across different types of critical information infrastructures?
6. Who should be in charge of operating critical information infrastructure?
7. What would be an example of a critical information infrastructure that performs functions within three different categories of critical information infrastructure functions?
8. How can critical information infrastructure operators avoid situations in which security mechanisms weaken privacy protections?
9. How can the boundary of a critical information infrastructure be identified?
10. What are effective management mechanisms that are capable of dealing with the challenges of CII operation?

References

- Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, Durumeric Z, Halderman JA, Invernizzi L, Kallitsis M (2017) Understanding the Mirai botnet. Paper presented at the 26th USENIX security symposium, Vancouver, BC, 16–18 Aug 2017
- AT&T (2010) AT&T completes 100-Gigabit Ethernet field trial. PR Newswire, 9 Mar 2010
- Ayres RU (1990) Technological transformations and long waves. Part I. Technol Forecast Soc Chang 37(1):1–37
- Bhagat S, Burke M, Diuk C, Filiz IO, Edunov S (2016) Three and a half degrees of separation. https://joytothehome.com/wp-content/uploads/2015/11/Three-and-a-half-degrees-of-separation_-_Blog_-_Research-at-Facebook.pdf. Accessed 15 Sept 2019
- Bondi AB (2000) Characteristics of scalability and their impact on performance. Paper presented at the 2nd international workshop on software and performance, Ottawa, ON, 17–20 Sept 2000
- Bostrom RP, Heinen JS (1977) MIS problems and failures: a socio-technical perspective. Part I: The causes. MIS Q 1(3):17–32

- Bye BL (2011) Volcanic eruptions: science and risk management. https://www.science20.com/planetbye/volcanic_eruptions_science_and_risk_management-79456. Accessed 15 Sept 2019
- Cadwalladr C, Graham-Harrison E (2018) Revealed: 50 Million facebook profiles harvested for Cambridge analytica in major data breach. *The Guardian*, 17 Mar 2018
- Caplan R, Donovan J, Hanson L, Matthews J (2018) Algorithmic accountability: a primer. https://datasociety.net/wp-content/uploads/2018/04/Data_Society_Algorithmic_Accountability_Primer_FINAL-4.pdf
- Carroll AB (1979) A three-dimensional conceptual model of corporate performance. *Acad Manag Rev* 4(4):497–505
- Carroll EC (2017) Making news: balancing newsworthiness and privacy in the age of algorithms. *Georgetown Law J* 106:69–114
- Clarke R (1999) Internet privacy concerns confirm the case for intervention. *Commun ACM* 42(2):60–67
- Cropanzano R, Byrne ZS, Bobocel DR, Rupp DE (2001) Moral virtues, fairness heuristics, social entities, and other denizens of organizational justice. *J Vocat Behav* 58(2):164–209
- Dehling T, Sunyaev A (2014) Secure provision of patient-centered health information technology services in public networks: leveraging security and privacy features provided by the German nationwide health information technology infrastructure. *Electron Mark* 24(2):89–99
- Dehling T, Gao F, Schneider S, Sunyaev A (2015) Exploring the far side of mobile health: information security and privacy of mobile health applications on iOS and android. *JMIR mHealth and uHealth* 3(1):e8
- Egan MJ (2007) Anticipating future vulnerability: defining characteristics of increasingly critical infrastructure-like systems. *J Conting Crisis Manag* 15(1):4–17
- Fekete A (2011) Common criteria for the assessment of critical infrastructures. *Int J Disaster Risk Sci* 2(1):15–24
- Gallagher R, Moltke H (2018) The NSA's hidden spy hubs in eight U.S. cities. <https://theintercept.com/2018/06/25/att-internet-nsa-spy-hubs/>. Accessed 15 Sept 2019
- Holwerda T (2011) DuckDuckGo: the privacy-centric alternative to Google. <https://www.osnews.com/story/24867/duckduckgo-the-privacy-centric-alternative-to-google/>. Accessed 15 Sept 2019
- Juiz C, Toomey M (2015) To govern IT, or not to govern IT? *Commun ACM* 58(2):58–64
- Kannengießer N, Lins S, Dehling T, Sunyaev A (2019) What does not fit can be made to fit! Trade-offs in distributed ledger technology designs. Paper presented at the 52nd Hawaii international conference on system sciences, Maui, HI, 8–11 Jan 2019
- Landau S (2015) Control use of data to protect privacy. *Science* 347(6221):504–506
- Laudon KC (1996) Markets and privacy. *Commun ACM* 39(9):92–104
- Martin K, Nissenbaum H (2016) Measuring privacy: an empirical test using context to expose confounding variables. *Columbia Sci Technol Law Rev* 18:176–218
- Meulen Rvd (2017) Gartner says 8.4 billion connected “things” will be in use in 2017, up 31 percent from 2016. <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>. Accessed 15 Sept 2019
- Nissenbaum H (1996) Accountability in a computerized society. *Sci Eng Ethics* 2(1):25–42
- Nissenbaum H (2010) Privacy in context: technology, policy, and the integrity of social life. Stanford University Press, Stanford, CA
- Oetzel MC, Spiekermann S (2014) A systematic methodology for privacy impact assessments: a design science approach. *Eur J Inf Syst* 23(2):126–150
- Oliver C (1991) Strategic responses to institutional processes. *Acad Manag Rev* 16(1):145–179
- Orlikowski WJ (2007) Sociomaterial practices: exploring technology at work. *Organ Stud* 28(9):1435–1448
- Orlikowski WJ, Scott SV (2008) Sociomateriality: challenging the separation of technology, work and organization. *Acad Manag Ann* 2(1):433–474

- Rinaldi SM, Peerenboom JP, Kelly TK (2001) Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE Control Syst Mag* 21(6):11–25
- Solove DJ (2002) Conceptualizing privacy. *California Law Rev* 90(4):1087–1155
- Sunyaev A, Huber MJ, Mauro C, Leimeister JM, Krcmar H (2008) Bewertung und Klassifikation von Bedrohungen im Umfeld der elektronischen Gesundheitskarte. Paper presented at the INFORMATIK 2008: Beherrschbare Systeme dank Informatik, Munich, 8–13 Sept 2008
- Travers J, Milgram S (1977) An experimental study of the small world problem. In: Leinhardt S (ed) *Social networks: a developing paradigm*. Academic Press, New York, NY, pp 179–197
- Trist E (1981) The evolution of socio-technical systems. In: *Perspectives in organization design and behavior*. Wiley, New York, NY, pp 32–47
- Union CotE (2008) Council directive 2008/114/EC on the identification and designation of European critical infrastructures and the assessment of the need to improve their protection. *Off J Eur Union L* 345(75)
- Warren SD, Brandeis LD (1890) The right to privacy. *Harvard Law Rev* 4(5):193–220
- Westin AF (1968) Privacy and freedom. *Washington Lee Law Rev* 25(1):166

Further Reading

- Clemons EK (2019) *New patterns of power and profit: a strategist's guide to competitive advantage in the age of digital transformation*, 1st edn. Palgrave Macmillan, Cham
- Dehling T, Sunyaev A (2014) Secure provision of patient-centered health information technology services in public networks: leveraging security and privacy features provided by the German nationwide health information technology infrastructure. *Electron Mark* 24(2):89–99
- Rinaldi SM, Peerenboom JP, Kelly TK (2001) Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE Control Syst Mag* 21(6):11–25

Chapter 12

Emerging Technologies



Abstract

Technological innovations have always played a key role in the human civilization's progression. Occasionally, these innovations develop to such an extent that they open up completely new pathways in individual fields and even reshape our society as a whole. However, to exert this prominent impact, a technology must first emerge, i.e. it must mature and become visible. This chapter introduces this process's underlying concepts and, thereby, provides a deep understanding of the emerging technologies' nature and their important implications, for example, affecting specific domains by changing the actors' and institutions' composition in this domain. To this end, this chapter gives an overview of the five most important attributes that characterize emerging technologies. The chapter also presents selected examples of currently emerging technologies, namely immersive technologies, virtual assistants, and artificial intelligence. Each technology is briefly described in accordance with its attributes, followed by a discussion of the technology's exemplary applications to highlight its emerging character, and, thus, provide an understanding of their potential future impact on specific application areas and our society as a whole.

The Learning Objectives of this Chapter

This chapter's main learning objective is to help readers understand the nature of emerging technologies and their important implications, for example, affecting specific domains by changing the composition of actors and institutions in this domain. After having studied this chapter, readers will be able to grasp the underlying concepts and distinguish clearly between established and emerging technologies in conceptual and applicational terms. First, from a conceptual perspective, studying this chapter will help readers understand the principle definition of emergence in the technology context and it will also help them become familiar with the five key attributes that characterize emerging technologies. Second, from an applicational perspective, this chapter will provide a practical understanding of technologies that are currently viewed as emerging. These technologies include immersive technologies, i.e. virtual reality and augmented reality, virtual assistants, and artificial intelligence. Based on these examples, readers will be introduced to these technologies' fundamental principles, learn the specific characteristics that make them emergent, and generally understand these technologies' potential impact on individual domains and/or society once they have fully emerged.

The Structure of this Chapter

The remainder of this chapter is structured as follows: First, the concepts of emergence and emerging technologies are introduced. Consequently, Section 12.1 discusses different definitions for the concept of emerging technology and presents five key defining attributes. Section 12.2 explains the first of three emerging technology examples explicated in this chapter, i.e. immersive technologies. Section 12.2 specifically explains mixed reality, augmented reality, and virtual reality by discussing their basic working principles, exemplary applications, and potential future impact. Similarly, the next two sections elaborate on two additional examples of emerging technologies, namely virtual assistants (Section 12.3) and artificial intelligence (Section 12.4). Finally, this chapter concludes with a short summary of the most important points.

12.1 Emergence and Emerging Technology

The pursuit of innovation in the form of technological advances has always been one of the driving forces behind humankind's progress. A few of these advancements stem from theoretical research, while others are based on practical developments or resulted from fortuitous discoveries. In certain cases, the resulting technologies are so radically new that they open up completely new avenues in particular fields or even spark large-scale societal change. The current candidates for such technologies are, for instance, immersive technologies, virtual assistants, and artificial intelligence (Gartner 2018). Owing to their prospective impact, these technologies are labeled as *emergent*. In general, the term emergent or emerge can be defined as “the process of coming into being, or of becoming important and prominent” (Stevenson and Lindberg 2010). Alternative definitions for *emergence* highlight different aspects of the concept, as exemplified in Table 12.1. Despite these differences, a common theme in these definitions is the verb "becoming", i.e. coming into existence. In this sense, *emergent* does not refer to a static thing – in the present case, a technology, – but to a thing that is involved in a process of change. The following key attributes describe this process's outcomes: visible, evident, important, and prominent (Rotolo et al. 2015).

Table 12.1 Definitions of the Concept Emergence (Rotolo et al. 2015)

Definition	Reference
“the process of coming into being, or of becoming <u>important</u> and <u>prominent</u> ”	(Stevenson and Lindberg 2010)
“to become manifest: become known [...]”	(Arthur 2007)
“to rise up or come forth [...] to become <u>evident</u> [...] to come into existence”	(Chao and Zhao 2013)
“move out of something and become <u>visible</u> [...] come into existence or greater prominence [...] become known [...] in the process of coming into being or prominence”	(Devezas 2005)
“starting to exist or to become known [...] to appear by coming out of something or out from behind something”	(Press 2019)

The ambiguity of the term *emergence* also affects the definition of *emerging technology*, which consequently is similarly diverse. As shown in Table 12.2, a few

of the definitions overlap, while others refer to entirely different aspects of the concept (Rotolo et al. 2015). For example, certain definitions claim that emerging technologies affect the economy and society (Porter et al. 2004; Martin 1995). Other definitions merely describe an emerging technology as being uncertain or non-specific and, thus, in a state of transition or change (Boon and Moors 2008; Halaweh 2013; Cozzens et al. 2010); and still other definitions merely name attributes, such as novelty and growth (Small et al. 2014; Cozzens et al. 2010). Yet, when synthesizing the different definitions, as Rotolo et al. (2015) did, five key attributes of an emerging technology can be singled out: (1) radical novelty, (2) relatively fast growth, (3) coherence, (4) prominent impact, and (5) uncertainty and ambiguity. These five attributes capture the emerging technology concept's essence and, thus, allow an understanding of what actually constitutes an emerging technology and how it differs from established technologies. Since it is crucial to fully comprehend these key attributes as this chapter progresses, the attributes will be discussed in more detail in the following paragraphs. Moreover, a corresponding short definition for each key attribute is provided in Table 12.3.

Table 12.2 Exemplary Definitions for the Concept of Emerging Technology, adapted from Rotolo et al. (2015)

Definition	Reference
“A ‘generic emerging technology’ is defined [...] as a technology the exploitation of which will yield benefits for a wide range of sectors of the economy and/or society”	(Martin 1995)
“[...] emerging technologies as science-based innovation that have the potential to create a new industry or transform an existing one. They include discontinuous innovations derived from radical innovations [...] as well as more evolutionary technologies formed by the convergence of previously separate research streams”	(Day and Schoemaker 2000)
“Emerging technologies are technologies in an early phase of development. This implies that several aspects, such as the characteristics of the technology and its context of use or the configuration of the actor network and their related roles are still uncertain and non-specific”	(Boon and Moors 2008)
“Emerging technology – a technology that shows high potential but hasn’t demonstrated its value or settled down into any kind of consensus. [...] The concepts reflected in the definitions of emerging technologies, however, can be summarised four-fold as follows: (1) fast recent growth; (2) in the process of transition and/or change; (3) market or economic potential that is not exploited fully yet; (4) increasingly science-based.”	(Cozzens et al. 2010)
“[...] emerging technologies are defined as those technologies that have the potential to gain social relevance within the next 10 to 15 years. This means that they are currently at an early stage of their development process. At the same time, they have already moved beyond the purely conceptual stage. [...] Despite this, these emerging technologies are not yet clearly defined. Their exact forms, capabilities, constraints, and uses are still in flux”	(Stahl 2011)
“Technical emergence is the phase during which a concept or construct is adopted and iterated by [...] members of an expert community of practice, resulting in a fundamental change in (or significant extension of) human understanding or capability.”	(Alexander et al. 2012)
“[...] there is nearly universal agreement on two properties associated with emergence – novelty (or newness) and growth.”	(Small et al. 2014)

The first attribute that classifies a technology as emerging is its *radical novelty*. An emerging technology breaks with present conventions by introducing a novel or discontinuous innovation (Small et al. 2014), which can refer to “a new idea, creative thoughts, new imaginations in form of device or method” (Merriam-Webster 2019b). Generally, technologies are radically novel when they fulfill a given function by using a different basic principle, i.e. the idea of using a certain central effect, compared to what was used previously to achieve the same or a similar function (Rotolo et al. 2015; Arthur 2007). A recent example for a radically novel technology can be found in the home entertainment industry, where online streaming is swiftly replacing most other forms of video content delivery, for example, TV broadcasts, DVDs, or video rental services (Chao and Zhao 2013). Following the evolutionary theory of technological change (Devezas 2005), novelty can also be achieved by introducing existing technologies to new contexts, i.e. other domains. This transfer of existing technology from one domain to another, a purported niche, is referred to as the specification process of technology. Such a niche differs from the domain of origin, i.e. where the technology was initially applied, in terms of adaptation, i.e. the needs of the niche and the abundance of resources. Analogous to Darwin’s evolutionary theory, applying a known technology concept to a potential niche can lead to adapting the technology to the needs of a niche and may be followed by the technology’s emergence within the specific context, i.e. filling out the niche. Relatively speaking, the technology can be understood as radically new in domains that differ from those where the technology was initially developed. Even though such evolutionary technologies are distinguishable from revolutionary technologies, i.e. technologies with relatively limited prior developments like DNA sequencing technologies. Evolutionary and revolutionary technologies represent a radical novelty in at least one context and, thus, point toward a technology’s emerging nature (Rotolo et al. 2015; Adner and Levinthal 2002).

The second attribute that classifies technology as emerging is *relatively fast growth*. The emerging technologies’ expansion rate usually exceeds that of established technologies (Rotolo et al. 2015; Cozzens et al. 2010; Small et al. 2014). Technology growth rates can be quantified via different indicators, like the number of actors involved in a technology’s advancement, e.g. researchers, practitioners, and politics, the amount of funding from public and private sources, the volume of knowledge output, e.g. publications and patents, or the number of newly developed prototypes, products, and services. Distributed ledger technology (DLT) is a prime example of an emerging technology with relatively fast growth, as the overall investment volume in companies associated with the technology, such as specialized hardware manufacturers, increases at exceptionally high speed (KPMG International 2018). Similar to the *radical novelty* attribute, a certain technology’s growth must be viewed in context. Accordingly, this technology may show an overall slow growth rate, while simultaneously exhibiting rapid growth compared to other technologies in its domain. Since both absolute and contextual growth rates can indicate the emergence of this technology, the attribute *fast growth* is preceded by the word “relatively” (Rotolo et al. 2015).

The third attribute that classifies technology as emerging is *coherence*. Emerging technologies have usually already adopted a coherent structure and, thus, must be distinguished from technologies that are still highly volatile and difficult to understand (Rotolo et al. 2015). This is because emerging technologies have surpassed the purely conceptual phase (Stahl 2011) and entered a stage defined by expansion and application that is fueled by its growing coherence, for example, through unification and standardization (Alexander et al. 2012; Rotolo et al. 2015). The emerging technology's coherence can be characterized by its terminological maturity, e.g. standardization of technical terms, abbreviations, and acronyms, (Reardon 2014), emerging scientific communication (e.g., formation of specialized conferences or journals) (Leydesdorff et al. 1994), and the existence of expert communities, which adopt and iterate the concepts or constructs underlying the particular emerging technology (Rotolo et al. 2015).

The fourth attribute that classifies a technology as emerging is *prominent impact*. Emerging technologies may affect specific domains or an entire socio-economic system by changing the composition of actors, institutions, patterns of interactions among these, and the associated knowledge production processes (Rotolo et al. 2015). Especially from an economic perspective, such technologies promise to affect a wide range of sectors (Martin 1995), give rise to entirely new industries, or fundamentally change existing industries, for instance, by altering the competition's rules or by changing the very nature of provided products or services (Day and Schoemaker 2000; Hung and Chu 2006). However, it is important to note that the emerging technology's prominent impact lies in the future. This means that an emerging technology has not yet reached its full potential, even though it is believed to hold an auspicious value proposition (Rotolo et al. 2015).

A widely known and often used indicator for such value expectations is the self-styled *hype cycle for emerging technologies*, which is created annually by the global research and advisory firm Gartner Inc.¹. As depicted in Fig. 12.1, the hype cycle consists of five successive phases that represent the emerging technology's increasing maturity level (Fenn and Blosch 2018). Following the hype cycle, expectations vary depending on the technology's maturity. At the beginning of the cycle, market expectations start to increase rapidly with early media interest in the potentially still unviable technology (*Innovation Trigger* phase). The expectation hype reaches its peak when the first successful applications of the technology become public and suggest high-value propositions (*Peak of Inflated Expectations* phase). With the number of technology implementations rising, the number of failed applications eventually increases as well, which leads to disenchantment and a waning interest in the technology (*Trough of Disillusionment* phase). As time passes, understanding the technology and how it is best applied grows, which leads to better implementations and again increases expectations (*Slope of Enlightenment* phase). Once a technology has matured enough for broad market applications, market expectations stabilize at an objective level (*Plateau of Productivity* phase). Knowing the emerging

¹<https://www.gartner.com>

technology's position within the hype cycle, therefore, allows assessing whether current market expectations are stable or not, which makes the hype cycle a good indicator when, for instance, deciding on technology investments (Fenn and Blosch 2018).

In the upcoming sections, this chapter will describe the following three technologies that are currently emerging and that can also be found within the depicted 2018 hype cycle: immersive technologies (Section 12.2), virtual assistants (Section 12.3), and artificial intelligence (Section 12.4).

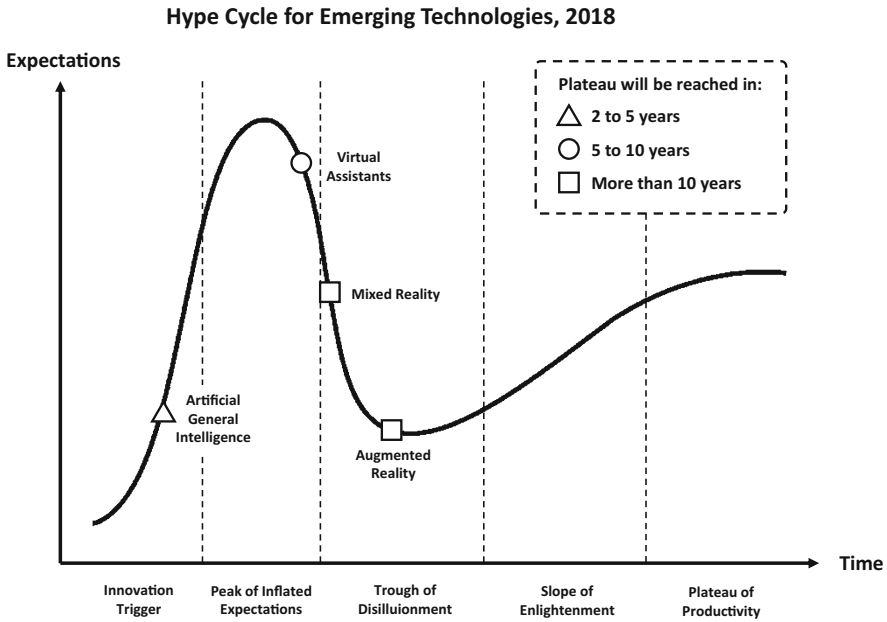


Fig. 12.1 Gartner Hype Cycle (adapted from Gartner (2017))

The fifth and final attribute that classifies a technology as emerging is *uncertainty and ambiguity*. Emerging technologies are generally less well understood, due to their novelty, which tends to make them unpredictable. These technologies are, therefore, characterized by ambiguity that often results from the different meanings social groups associate with the emerging technology (Stirling 2007; Mitchell 2007). A possible sign of ambiguity is diverging visions of a technology's future, which can be found, for instance, in news articles, editorials, and perspective articles that are published in professional and scholarly journals (Rotolo et al. 2015). Ambiguity also results from incomplete knowledge of the emergence's possible outcomes (Mitchell 2007; Rotolo et al. 2015), which, in turn, creates uncertainty. In this context, uncertainty relates to a technology's outcomes, – for example, potential applications of the technology, financial support for its development, standards, and production

costs, – and utilization, which could be either intended or unintended, as well as desirable or undesirable (Stirling 2007; Rotolo et al. 2015; Boon and Moors 2008).

Emerging Technology
An emerging technology is a radically novel and relatively fast-growing technology characterized by a certain degree of coherence persisting over time and with the potential to exert a considerable impact on the socio-economic domain(s). This impact is observed in terms of the composition of actors, institutions, and patterns of interactions among these, along with the associated knowledge production processes. An emerging technology's most prominent impact, however, lies in the future and is, therefore, still somewhat uncertain and ambiguous in the emergence phase (Rotolo et al. 2015).

Table 12.3 The Emerging Technologies' Five Key Attributes

Attribute	Definition
Radical novelty	<i>Radical novelty</i> refers to an emerging technology’s ability to fulfill a given function by using a radically different basic principle, i.e. the idea of using a certain central effect, compared to what was used previously to achieve a similar purpose (Rotolo et al. 2015; Arthur 2007; Day and Schoemaker 2000; Small et al. 2014).
Relatively fast growth	<i>Relatively fast growth</i> refers to the rapid expansion of emerging technologies that can be assessed through different indicators, – for example, the number of actors involved, the knowledge outcomes, or the funding volume, – and is usually evaluated in comparison with other technologies in their particular domain (Cozzens et al. 2010; Small et al. 2014).
Coherence	<i>Coherence</i> refers to a technology’s inherent unification and standardization that, for instance, is signaled by its terminological maturity, emerging scientific communication, and the existence of expert communities, which adopt and iterate the concepts or constructs underlying the particular emerging technology (Day and Schoemaker 2000; Stahl 2011; Alexander et al. 2012; Reardon 2014; Cozzens et al. 2010).
Prominent impact	<i>Prominent impact</i> refers to the emerging technologies' potential to affect specific domains or an entire socio-economic system by changing the composition of actors, institutions, patterns of interactions among these, and the associated knowledge production processes, although the full manifestation of this impact lies in the future (Rotolo et al. 2015; Martin 1995; Day and Schoemaker 2000; Porter et al. 2004; Corrocher et al. 2003; Hung and Chu 2006; Cozzens et al. 2010; Stahl 2011; Alexander et al. 2012).
Uncertainty and ambiguity	<i>Uncertainty and Ambiguity</i> refers to an emerging technology’s incalculable outcomes, – for example, potential applications of the technology, financial support for its development, standards, and production costs, – and unforeseeable applications, as well as the different meanings social groups associate with the technology (Stirling 2007; Rotolo et al. 2015; Day and Schoemaker 2000; Porter et al. 2004; Hung and Chu 2006; Boon and Moors 2008; Cozzens et al. 2010; Stahl 2011; Halaweh 2013).

The emerging technologies' five key attributes cover a spectrum ranging from low to high levels. Especially *radical novelty* and *relatively fast growth* are not absolute, but relative, terms. They might indicate the emergence of a technology if they are applied in a certain domain in which the technology is likely to emerge, and if they are compared to established technologies in this domain (Rotolo et al. 2015). Please note that the five key attributes are independent emergence indicators. Consequently, the same technology will, at a particular stage of emergence, most likely show a different level for each of the different attributes, for example, the attributes may evolve slowly or rapidly, or even at the same time. As depicted in Fig. 12.2, the following three stages of emergence can be distinguished, namely (1) pre-emergence, (2) emergence, and (3) post-emergence (Rotolo et al. 2015).

In the pre-emergence stage, i.e. the earliest stage of emergence, a technology has recently entered into a certain domain, which usually leads to high levels of *radical novelty*, as well as *uncertainty* and *ambiguity* compared to other technologies in the same domain. The technology also shows low levels of *coherence*, e.g. different expert communities are involved in technology development, *prominent impact*, e.g. only a few people can already assess its potential impact, and *growth*, e.g. applications for the technology are still unknown. Since the technology has not yet started to grow and since it is associated with high levels of uncertainty, its future remains unclear at this stage and it might never emerge.

However, if the technology progresses and a dominant expert community forms, the technology becomes more coherent and its impact becomes less uncertain and ambiguous. As a result, the technology begins to settle in terms of publications, patents, researchers, firms, prototypes/products, and so forth. At the same time, the technology's radical novelty is likely to decrease. Other technologies using different basic principles are also likely to emerge in the same domain in which the technology is emerging (Rotolo et al. 2015). The technology, thereby, enters the emergence stages in which the attributes change dramatically and the technology can be referred to as *emergent*. Attributes, such as impact and growth, might reach a stable or even a declining level. The technology is no longer termed as radically novel, knowledge of the technology's possible outcomes becomes more complete, and expert communities in research and practice have been established (e.g., regular conferences taking place).

Finally, and on its way to become established, the technology enters the purported post-emergence stage, i.e. a late phase of emergence. During this stage, all five key attributes start to stabilize at their individual high or low point (Rotolo et al. 2015). Fig. 12.2 shows how the attributes change, along with the three stages, i.e. pre-emergence, emergence, and post-emergence. The changing levels of the attributes named relatively fast growth, coherence, and prominent impact, are depicted as an S-curve, whereas the changing levels of the attributes named radical novelty, and uncertainty and ambiguity, are illustrated as a reverse S-curve.

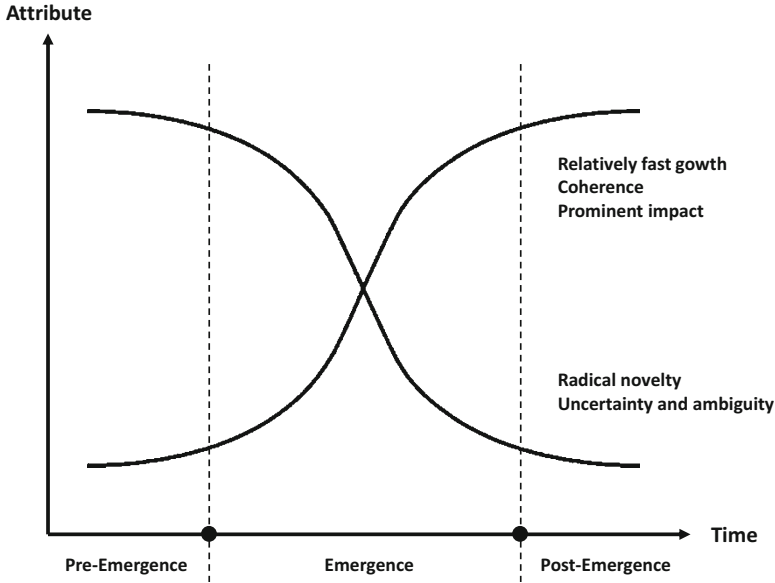


Fig. 12.2 Pre-Emergence, Emergence, and Post-Emergence (adapted from Rotolo et al. (2015))

Next, immersive technologies are briefly summarized. Referring to Gartner’s hype cycle (Fig. 12.1), immersive technologies have already passed the peak of inflated expectations and are heading for the trough of disillusionment. These technologies will probably reach the plateau in five to ten years.

12.2 Immersive Technologies

An important concept that spawned several emerging technologies in different contexts is *immersive technology*. The concept subsumes different technological approaches for generating computer-based simulations of reality with physical, spatial, and visual dimensions, and thereby blurs “the lines between the physical and virtual worlds, creating a sense of immersion and enhancing the realism of virtual experiences” (Suh and Prophet 2018, p. 78). This sense of immersion is created through temporarily changing a person’s sense of presence by tricking her/his cognitive and perceptual systems into believing that she/he is somewhere other than her/his actual physical location (Brown and Cairns 2004).

The degree of immersion may vary with different users and applications. Players’ involvement in computer games is a good example of understanding these variations, as they are able to immerse themselves to varying degrees in artificially generated worlds (Brown and Cairns 2004; Jennett et al. 2008). Here, three levels

of involvement are distinguishable, namely *engagement*, e.g. invest time, effort, and attention in the game, *engrossment*, e.g. invest a high level of emotion in the game), and *total immersion*, e.g. fully present and empathize with the game.

Immersive Technologies
Immersive technologies generate computer-based simulations of reality with physical, spatial, and visual dimensions that create a sense of immersion and enhancing the virtual experiences' realism (Soares and Simão 2019; Suh and Prophet 2018).

The concept of immersive technologies is an overarching concept and includes categories like virtual reality (VR), augmented reality (AR), and mixed reality (MR). VR creates a fully enclosed non-physical world, which is intended to be experienced as being physically present through, for instance, using head-mounted displays (Freina and Canessa 2015). MR combines real and virtual contents, using digital devices, such as a smartphone, and is, therefore, viewed as consisting of AR and augmented virtuality (AV). Whereas AR refers to virtual 3D objects in immersive reality, AV is the converse case, meaning that AV captures features of reality in immersive virtual 3D environments. Fig. 12.3 shows a reality–virtuality continuum in which the real environment, or reality, and a virtual environment are shown on opposite sides of the continuum. MR, AR, and AV are in between the two extremes (Milgram and Kishino 1994).

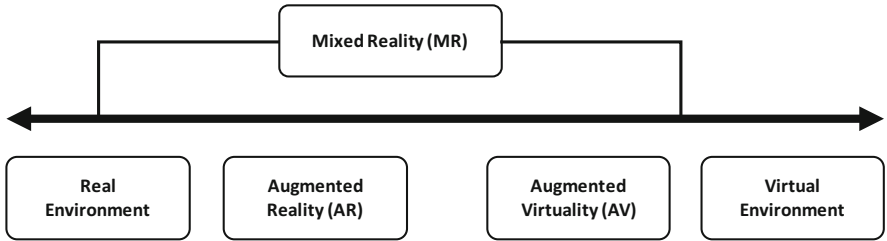


Fig. 12.3 Reality–Virtuality Continuum (adapted from Milgram and Kishino (1994))

As explained in the following two subchapters, especially VR and AR, along with their application in certain contexts, are perceived as emerging technologies. They are expected to have a disruptive impact on existing contexts, markets, and applications, similar to the effects that followed the introduction of personal computers and smartphones (Parvinen et al. 2019). A prime example of this impact is the healthcare context in which the very few applications of immersive technologies have already gained worldwide attention, and numerous other mainstream applications in hospitals and primary care may greatly benefit via the employment of AR and VR (The Medical Futurist 2016a; The Guardian 2016; Pillay 2018).

12.2.1 Virtual Reality

In 1988, Jaron Lanier coined the term virtual reality (VR) (Luciani 2007), even though the technology's roots can be traced back to the works of Ivan Sutherland who established the field of interactive computer graphics in the 1960s (Sutherland 1964). Based on Jaron Lanier's research, the first wave of VR systems emerged in the early 1990s. This wave was marked by the tentative application of VR technology in a variety of different fields, which were inspired by advances made in the computing gaming industry. However, the then state-of-the-art computer, sensor, and display technologies were still too limited and expensive, leading to unsatisfying user experiences and fast declining interest in the technology (Botella et al. 1998; Parvinen et al. 2019). The second VR wave started in the mid-2000s. Based on the technical advances made over the past decade, the technology became more interesting for a wide array of applications in fields, such as engineering, healthcare, construction, education, entertainment, retail, marketing, and military (Parvinen et al. 2019).

VR can be defined as a “technology that generates an interactive virtual environment that is designed to simulate a real-life experience” (Suh and Prophet 2018, p. 78). Users can experience and explore these virtual environments interactively. Interactivity, in this context, refers to “the degree to which users of a medium can influence the form or content of the mediated environment” (Steuer 1992, p. 80). This means that users can, for instance, create new objects, move through the virtually created space, change perspectives, or manipulate other environmental parameters. The perception of the virtual environment is predominantly created through visual stimulation, but can also involve audio, tactile, and other forms of feedback (Handa et al. 2012). These stimuli's main goal is to convince users to feel as if they are physically present in a particular environment. Since this perception is only virtual, *presence* in the VR context can be understood as “the subjective experience of being in one place or environment, even when one is physically situated in another” (Witmer and Singer 1998, p. 225).

Virtual Reality

Virtual Reality refers to technology that generates an interactive virtual environment that is designed to simulate a real-life experience (Suh and Prophet 2018).

VR can be differentiated into non-immersive and immersive VR. Non-immersive VR presents virtual content on a normal computer screen without employing other means for increasing immersion. Non-immersive VR can usually be found on desktop computers, smartphones, and video game consoles. Interactions within the created virtual environments, which can be two-dimensional or three-dimensional, are usually implemented with traditional input devices, like a keyboard, mouse, or

game controller (Handa et al. 2012). A good example of non-immersive VR is web-based virtual environments, such as Second Life (Suh and Prophet 2018).

Immersive VR, in contrast, provides richer modes of interaction by utilizing complex sensors and visualization devices, as depicted in Fig. 12.4. A common component used for this type of VR are head-mounted displays (HMDs) with motion tracking capabilities that change the user's perspective in the virtual environment according to her/his physical motions. Similar to devices used in AR applications, HMDs for VR are basically helmets or smart glasses holding one or two LCD (liquid crystal display) screens in front of the user's eyes, which provide a (stereoscopic) view of the virtual environment (Handa et al. 2012). Another approach for creating an immersive VR experience is the purported *cave automatic virtual environment* (CAVE). In a CAVE, the virtual world's depiction is projected onto the walls of a room-sized cube in which a user can stand or freely move around, often without the need for a HMD (Suh and Prophet 2018). This enables a more focused and naturalistic interaction with the computer that goes far beyond the point and click interface (Handa et al. 2012). This means that the level of immersion is drastically increased compared to those of non-immersive VR or traditional computer interactions (Suh and Prophet 2018).

These recent advancements with regard to newly developed devices, components, software, and user interfaces are an indicator for the emergence of VR technology, i.e. rapid growth, and also pave the way for radically new applications and markets (Parvinen et al. 2019). As stated, these applications can be found in the healthcare sector where VR usage is emerging, but not yet widespread, except for a few selected cases and pilot projects. For example, in 2016 and for the first time in medical history, the cancer surgeon, Shafi Ahmed, performed an operation that was filmed by two 360-degree cameras. People all over the world could watch the operation in real time via the Medical Realities website² and within a rendered VR environment³ (The Medical Futurist 2016a; The Guardian 2016). This event marked an important milestone, particularly for medical students of whom only a few can peek over the surgeon's shoulder during an operation or a surgical treatment. Medical students can, therefore, benefit greatly from VR during their training by allowing them, with the help of HMDs, to virtually be present in the operation room. Another good example of a pilot project in the healthcare context is "We Are Alfred", lead by Embodied Labs⁴. The project aims at teaching medical students and students of other health professions to be more empathetic with elderly people. In the self-styled Alfred Lab, students can experience how it feels to be Alfred, a 74-year-old man with audio-visual impairments. Alfred's disabilities are simulated with the help of an HMD, headphones, and hand-tracking devices. The basic idea behind this experience is that empathy between caregivers (i.e., physicians, nurses, etc.) and patients can be fostered easier when caregivers have actually seen and experienced a few of the

²<https://www.medicalrealities.com/>

³<http://vlippmed.com/>

⁴<https://embodiedlabs.com/>

age-related impairments their patients suffer from (The Medical Futurist 2016a; Gaines 2016; Dyer et al. 2018).

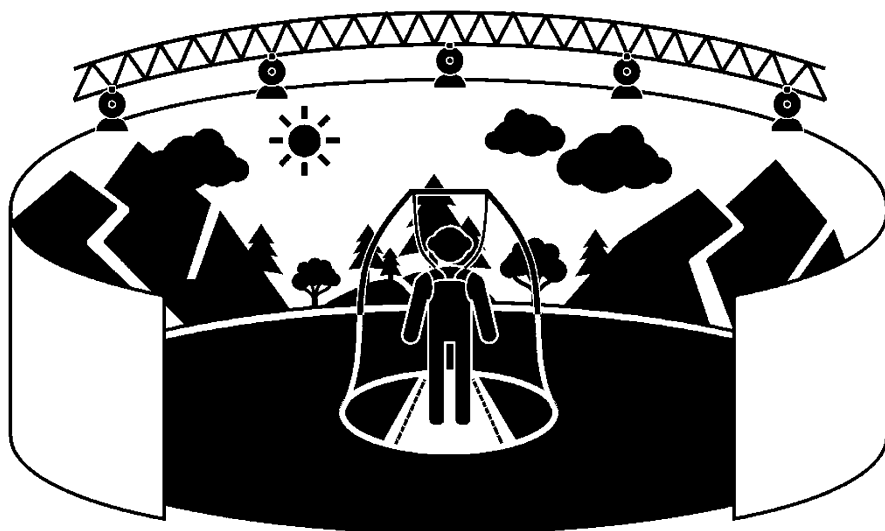


Fig. 12.4 Example of a Virtual Reality Room Setup

Other pilot projects demonstrate how patients can use VR technology directly. For example, Brennan Spiegel and his team at the Cedars-Sinai Hospital in Los Angeles introduced a VR system that is used in pain therapy. The system, which is called First Hand Technology⁵, allows patients to escape the hospital room and explore an artificial world to distract themselves from their pains. The initial promising results from the project show that, via VR technology, hospitalization time and the amount of resources utilized decrease, owing to the reduction of patients' stress and pain. A similar VR-based project named Farmoo⁶ intends to support teen cancer patients in becoming distracted during chemotherapy treatments. The idea is that the patients concentrate on the activities within a game instead of only thinking of their illness and the treatment (The Medical Futurist 2016a; The Peak 2016; Pillay 2018).

Nonetheless, despite these and many other potential beneficial applications of VR, the technology suffers from a few prevailing issues that create uncertainty and that must, thus, be addressed in order to promote its widespread adoption. Especially the last examples demonstrate the use of total immersion in a virtual environment to create the desired effect, for example, distraction. However, it has been shown that

⁵<https://firsthand.com/>

⁶<https://hwlo.ca/farmoo.html>

deep immersion bears the risk of losing touch with reality and lingering longer in these environments (Ortiz de Gortari and Griffiths 2015). It is also possible that the mainstream use of VR headsets will cause an increase in the occurrence of the purported game transfer phenomena (GTP). In this condition, a VR user's brain starts to confuse virtual and physical events. GTP manifests via symptoms, such as seeing objects turn into clusters of pixels, hearing sounds from games when one falls asleep, or even swerving to avoid Battlefield-style land mines on the highway. While GTP is currently an issue predominantly observable in the computer gaming context (Ortiz de Gortari and Griffiths 2015), it might become a more widespread phenomenon with the emergence of immersive technologies. Finally, VR technology still struggles with issues that cause a number of users to experience visually-induced motion sickness (Hettinger and Riccio 1991; Mazloumi et al. 2018). The resulting symptoms include, for example, general discomfort, nausea, disorientation, and apathy (Kolasinski 1995). Visually-induced motion sickness is often caused by a user's perception of self-motion within the virtual environment, such as a walking animation from a first-person perspective, without correlating physical movements. To mitigate this problem, VR application developers must gain a better understanding of the perception's psychological effects. Visually-induced motion sickness is also caused by prevailing insufficiencies in the VR technology, like imprecise physical motion detection, display delays, and issues related to the field of view and viewing angle (Ruddle 2004; Groen and Bos 2008). The lack of understanding the VR technology's psychological effects, as well as the prevailing technical issues, shows that VR is surrounded by uncertainty and emphasizes that the technology is still emerging.

12.2.2 *Augmented Reality*

With regard to the reality–virtuality continuum (see Fig. 12.3), MR is somewhere in between the real and the virtual environment's extrema. MR also concerns AR (Milgram and Kishino 1994). The basic idea behind AR, i.e. blending the real and the virtual, dates back to the 1950s when cinematographers started to use virtual objects in movies, for example, actors/actresses standing in front of blue screens (Zahedi et al. 2016). AR overlays the world in front of users with virtual objects (i.e., computer-generated annotations), thereby augmenting their reality (Merriam-Webster 2019a) (Fig. 12.5). AR enhances the user's perception of and interaction with the real world (Carmigniani and Furht 2011). In addition to augmenting reality, AR applications are characterized by two properties, namely running in real-time, i.e. the overlay is displayed with little to no temporal delay, and aligning virtual objects and real-world structures, i.e. creating a contextual relation (Azuma 1997). Conversely, objects can also be removed from a real environment by covering them with virtual information. The approach of hiding certain real-world objects behind virtual objects with the intention to eliminate them from a user's perception is called *mediated* or *diminished reality* (Azuma et al. 2001).

Beyond this short introduction, the technologies, examples, etc., presented next will also apply to the broader MR field, defined as the space in which the physical and virtual worlds co-exist (Suh and Prophet 2018). When virtual objects are added to the perception of the real environment, they often convey information that users would not be able to detect with their natural senses. An example is an AR device that shows construction workers the exact position for drilling a hole into a wall without damaging the wiring. Occasionally, this information can also have an entertainment purpose (Carmigniani and Furht 2011), as in the case of Pokémon Go. In 2016, the mobile game, Pokémon Go, was released and became famous for incorporating technology into the story within the game. In Pokémon Go, a player's real environment is overlaid with virtual in-game objects and characters, which require users to physically move to certain geographical points, for example, to catch Pokémon characters or to acquire game items by visiting Pokéstops and Gyms (Hino et al. 2019; Althoff et al. 2016).

These examples already hint that mobility is a very important factor when considering most AR applications. Hence, one of the main enablers for the emergence of AR is the recent advancements made in mobile computing and telecommunication infrastructure (Sommerauer and Müller 2018). These advancements lead, most importantly, to a higher performance and miniaturization of the essential hardware components that are necessary for creating an augmented reality, namely (1) devices for displaying the virtual objects, (2) input devices to detect users' interaction with the real and virtual objects, (3) sensors to capture users' positions and movements, and (4) computing devices that combine the input data and create the virtual overlays.

With regard to display devices, the most commonly used variants are HMDs, handheld devices, and spatial displays. As the name suggests, *HMDs* are displays worn on the head as a helmet or eyeglasses. These displays place the virtual overlay between the users' eyes and their view of the world. *Handheld devices*, like smartphones or tablet PCs, can also create AR experiences. However, unlike HMDs like Google Glass, *handheld devices* usually do not have a translucent display. This means that the reality is captured by the device's camera, augmented with additional objects, and displayed over the built-in display.⁷ *Spatial displays* are not actual displays, but projectors that display virtual objects directly on physical objects. Consequently, users do not need to wear or hold any device, which allows for more natural forms of interaction with objects and collaborations between users (Raskar and Low 2001).

Besides displays, AR systems can utilize different types of input devices to detect users' interactions with the physical or virtual world, like gloves, wristbands, and smartphones. In order to detect a user's motions and movements, tracking devices like cameras, GPS receivers, accelerometers, and compasses are used. Each of these technologies has different levels of accuracy and is either stationary (i.e., observe the user) or carried by the user (i.e., observe the environment's relative movement).

⁷Note that this is not the case in MR when using HoloLens.

Finally, AR systems require computational resources to process the sensor data and create a corresponding overlay. In the past, AR often required users to carry a laptop, a backpack, or a wired connection to a nearby stationary system (Carmigniani and Furht 2011). Modern AR systems usually employ more mobile computing devices, such as smartphones and tablets.

Augmented Reality

Augmented reality describes an enhanced version of reality that is created by using technology to overlay digital information on an image of a thing, which is viewed through a device, such as a smartphone camera (Merriam-Webster 2019a).

The emerging character of AR becomes clear, particularly when considering the number of innovative services that are currently created in areas like marketing, entertainment, education, and healthcare (Carmigniani and Furht 2011). For instance, AR technology is used for enhancing consumers' shopping experience (de Regt and Barnes 2019), as in the case of the IKEA Place app, IKEA Place, which allows a consumer to first visualize products, such as furniture, in her/his home before buying them. The app allows consumers to virtually redecorate any room in their home before purchasing the actual physical objects. Using the smartphone's camera, the app recognizes the room's dimensions and enables consumers to virtually manipulate the elements within it, such as adding or removing furniture, changing the wall color, or putting down new flooring (IKEA 2017).

Health IT has the potential to improve the productivity and quality of healthcare services and therefore plays a crucial role in delivering innovative healthcare services (Mandl and Kohane 2017; Gao and Sunyaev 2019). Innovative healthcare services are another example that underlines the AR technologies' emergence character. In case of an emergency, AR can offer lifesaving contextual information to the people who provide first aid (The Medical Futurist 2016b; Pillay 2018). For example, with the help of a smartphone's camera, the app, AED4EU⁸, allows locating the nearest automated external defibrillators (AEDs). The exact position of the nearest AEDs is projected onto the user's current environment via their smartphones' screens. With recent advancements in wearable AR devices, especially smart glasses, the technology also facilitates the development of personalized medical treatment applications (de Regt and Barnes 2019). For example, the company, Small World, has started a project in collaboration with the Australian Breastfeeding Association in which AR smart glasses, i.e. Google Glass, are used to support new mothers who struggle with breastfeeding. Trained counselors can, so to speak, see through the eyes of the mothers while they are breastfeeding. The counselors can, thereby, guide mothers through the breastfeeding process and give them personalized advice in the form of visual step-by-step instructions (The Medical Futurist 2016b; Small World 2017; Joiner 2018).

⁸<https://www.layar.com/layers/sander1>

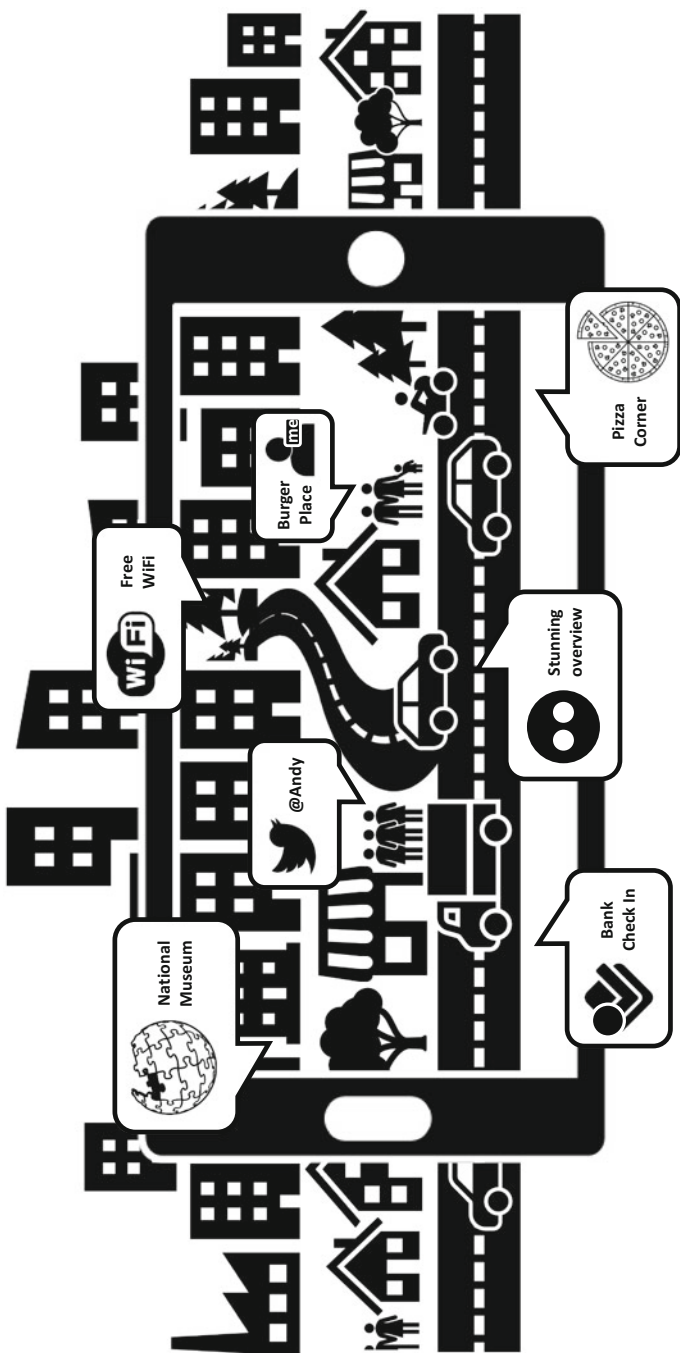


Fig. 12.5 Example of an Augmented Reality Application

The above mentioned examples show that AR has started to move from the laboratory into consumer markets (Daponte et al. 2014), but it is still too early to label the technology as established. Probably the clearest indication of this is that AR still faces a high level of uncertainty with regard to user acceptance, especially when used in the public space. This was clearly demonstrated with the failed introduction of Google Glass. Soon after its launch in 2013, users wearing Google Glass were regularly accused of invading the privacy of the people surrounding them, due to the front-facing camera mounted on the AR device. Google Glass wearers were even physically assaulted in a few cases (Eisenmann et al. 2014). Due to the resulting public discussion (Warman 2013), businesses started to ban Google Glass wearers from entering their premises (Kudina and Verbeek 2018). Even though Google Glass was discontinued in 2015, recent consumer surveys still show that using AR technology has a negative connotation, owing to privacy concerns (Rauschnabel et al. 2018). The emergence of IT in healthcare, e.g. AR, during recent years has led to intense involvement of physicians and patients with computer technology, e.g. HMDs. In general, user acceptance is an important factor for successful adoption and utilization of the targeted technology (Dünnebeil et al. 2012). Overcoming acceptance barriers, as in the case of Google Glass, is one of the biggest challenges that must be addressed before AR, as a technology, can finally emerge and realize its prominent impact.

A brief summary of virtual assistant is presented next. Referring to Gartner's hype cycle (Fig. 12.1), this technology has also passed the peak of inflated expectations and is heading for the trough of disillusionment. Virtual assistant will probably reach the plateau in two to five years.

12.3 Virtual Assistant

The digital age brings new possibilities to ease many activities for business and private purposes. As an emerging technology, virtual assistants, such as Google Assistant⁹, rely on other emerging technologies, for instance, natural language processing and artificial intelligence. Virtual assistants have recently reached the consumer market and are expected to become an integral part of our everyday lives (Sciuto et al. 2018). As an emerging technology, a virtual assistant is still a very ambiguous concept. This is evidenced, for example, by the technology's various labels used across different media, for example, academic literature, product descriptions, blog entries: intelligent personal assistants, voice assistants, smart personal assistants, digital personal assistants, or virtual personal assistants.

Generally, virtual assistants aim to provide personalized, digital assistance to human users. This idea can be traced back to the early days of handheld computers in the 1980s. Back then, the main purposes of purported personal digital assistant

⁹<https://assistant.google.com/>

(PDA) devices was to store information, such as contact data and calendar entries, and perform very simple tasks, like basic calculations. Well-known examples of PDAs are the Psion Organizer, Apple Newton, IBM Simon, and Palm Pilot. Users interacted with PDAs by physically pressing buttons or, if a touchscreen interface was provided, by dragging a finger or a stylus. In terms of assistance, PDAs were basically more convenient, digital versions of paper-based organizers. The PDAs' successor, the smartphone, vastly extended the PDA concept, especially regarding application variety and communication capabilities. However, concerning personal assistance, smartphones, until recently, still relied on interaction modes that are similar to those of PDAs, i.e. buttons or touchscreens, and required precise user commands for information input and retrieval.

In contrast, the now emerging virtual assistants utilize recent advances made in the field of AI to interpret human speech, autonomously execute the interpreted commands, and respond via synthesized voices (Hoy 2018). Users can, for example, verbally ask a virtual assistant “What’s my agenda for today?” instead of using a dedicated calendar app on their smartphones or computers to find the same information. This exemplifies two of the virtual assistants' most distinguishing characteristics: first, their natural form of user interaction and, second, autonomous task completion abilities. Current examples of virtual assistants include Apple’s Siri, Amazon’s Alexa, Microsoft’s Cortana, and the Google Assistant, all of which exhibit rapid growth rates regarding adoption and mainstream consumer acceptance (Lopatovska et al. 2018). A few of these virtual assistants are already embedded in smartphones or they run on more dedicated devices, such as smart speakers. Amazon’s Alexa, for instance, is used in the smart speakers Amazon Echo and Amazon Echo Dot. These devices allow for a broad range of new services, especially for private purposes. When integrated into the existing infrastructure of a private home, they can, for instance, carry out the following tasks: (1) send and read text messages, make phone calls, as well as send and read e-mail messages; (2) answer simple questions, e.g. “What time is it?” or even more tricky ones, e.g. “What do elves eat for Christmas?”; (3) set timers, alarms, and calendar entries; (4) set reminders, make lists, and do basic math calculations; (5) control media playback from connected services, such as Amazon, Google Play, iTunes, etc.; (6) control the thermostats, lights, alarms, and locks (Hoy 2018).

Virtual Assistant

Virtual assistants refer to software agents that are able to interpret human speech, autonomously execute the user-issued command, and respond via synthesized voices (Hoy 2018).

Besides using virtual assistants to increase convenience in the personal domain, another virtual assistant variant that should have a prominent impact in business and general public contexts is the self-styled *chatbot*. Chatbots are software agents that interactively communicate with human users via text or voice (Chung et al. 2017).

The chatbots' origins predate those of the virtual assistant concept by two decades. The first interactive computer program that is presently recognized as a chatbot was developed by Joseph Weizenbaum at the MIT Computer Science and Artificial Intelligence Laboratory in 1966 (Janarthanam 2017). The program named ELIZA was able to engage a human in conversation by imitating the answers a psychoanalyst might answer, paraphrasing the questions it was asked. However, the program was limited to very narrow ranges of questions (Fryer et al. 2019).

Extending the concept of virtual assistants, chatbots are designed to engage in a human-like, interactive conversation instead of merely understanding and executing commands. When provided as a service, for example, by a private company or public agency, chatbots are good at satisfying the customers' complex information needs. For instance, a conversation could be initiated by a customer's very unspecific and unstructured problem statement. A chatbot would then imitate the behavior of a service employee by stepwise narrowing down the issue to reach an understanding of the core problem. Based on this understanding, and similar to Alexa and the other above mentioned virtual assistants, the chatbot would then provide a suitable answer and, if required, initiate additional process steps. This shows that chatbots are good at complementing or even replacing existing direct and indirect communication channels, e.g. face-to-face, phone, e-mail, static texts, thereby, decreasing service costs while increasing service and information quality.

Chatbot

Chatbots refer to software agents that are designated for interactive communication with humans via text or voice (Chung et al. 2017).

Besides their general prospects, chatbots are considered to be an emerging technology, which, again, can be explicated in the healthcare context. There, it is envisaged that chatbots could make consultations with physicians obsolete in many cases, thus demonstrating their radical novelty. On the one hand, chatbots may help address diagnosable health concerns and, thus, unburden medical professionals, especially doctors in primary care, such as general practitioners (GPs). On the other hand, chatbots could support patients in managing their healthcare (e.g., medication management) and in taking more responsibility for their personal health (The Medical Futurist 2017a, 2018b). Generally, the vision for the future is that medical chatbots will become the initial contact point for patients in primary care. This means that patients will, at first, not be directly in touch with physicians or nurses. Patients will, instead, turn to medical chatbots to detail their symptoms and ask questions. If, based on a chatbot's diagnosis, further treatment steps are required, the patient will be directed to an appropriate medical expert. Nowadays, the number of medical chatbots increases at a fast pace (Zion Market Research 2019), thus demonstrating the technology's relatively fast growth. The following examples show the direction that current developments of medical chatbots take.

The medical app Ada¹⁰ can assess its users' health based on the indicated symptoms, using an AI-based database. In 2016, Ada was launched globally and became one of the world's fastest-growing medical apps by 2017. Since its launch, more than 1.5 million people have used the app. Ada's overall goal is to help patients better understand their health and navigate them to the appropriate care. In the provided chat interface, Ada allows users to investigate their complaints and also connects them with a physician. Ada enables users to share their health assessment with a physician or directly consult with a physician via Ada's Doctor Chat feature. Furthermore, Ada developers have experimented with a voice interface using smart speakers like Amazon's Alexa (The Medical Futurist 2018b; Ada 2019).

Babylon Health¹¹ takes this idea a step further in terms of interactivity. The platform offers AI-enabled chatbot consultations based on personal medical history and common medical knowledge, as well as live video consultation with a GP. In the first case, users report the symptoms of their illness to the Babylon Health app through a verbal or graphical interface. The app checks the symptoms against a database of diseases and offers an appropriate course of action. In the second case, GPs listen to users, i.e. patients, diagnose, and administer therapy, e.g. write prescriptions or refer the patient to a specialist physician if necessary. This case goes far beyond the general idea of a chatbot (Babylon 2019; The Medical Futurist 2018b). As stated, the basic idea of medical chatbots is to support the process of personalizing the patients' healthcare until a GP takes over, while having access to a pre-diagnosis. On the one hand, the extended approach of Babylon Health may be more convenient and trust-inducing for patients, as they do not need to solely rely on an AI-based diagnosis. On the other hand, the approach bears the risk of opportunistic behavior by patients, as shown in a test of the app in a pilot project. Instead of reducing GP attendance, as originally intended, the test raised the concern that patients may have used the app untruthfully in order to gain faster access to a GP appointment, for example, by reporting fake symptoms (Pulse 2017).

Nonetheless, in general, the emergence of virtual assistants in the form of medical chatbots promises to lighten the burden of medical professionals, especially regarding easily diagnosable health concerns or quickly solvable health management issues. Virtual assistants, therefore, are good at drastically decreasing costs to the healthcare system and increasing treatment quality by allowing medical professionals to focus on more critical or complicated treatment measures, thus, demonstrating the virtual assistants' prominent impact as an emergent technology for the healthcare system (The Medical Futurist 2017b). However, medical chatbots are still surrounded by uncertainty. As the field is still relatively young, medical chatbots are still prone to failure. The Babylon Health example clearly shows that chatbots are susceptible to deliberate manipulations, whether for the benefit or to the detriment of individual users or user groups. In conclusion, whereas medical chatbots are useful by giving quick responses, they will most likely not replace GPs, neither now nor in

¹⁰<https://ada.com/>

¹¹<https://www.babylonhealth.com/>

the near future. Once the technology has evolved further and finally emerged, this might change. However, for the time being, patients still need to verify the diagnoses of virtual assistants with their GPs before taking any actions.

A brief summary of artificial intelligence follows next. With regard to Gartner's hype cycle (Fig. 12.1), artificial intelligence refers to an innovation trigger and climbs up to the peak of inflated expectations. Artificial intelligence will probably reach the plateau in more than ten years.

12.4 Artificial Intelligence

Artificial intelligence (AI) as an academic discipline was founded in 1956. Since then, the discipline underwent striking developments. Simply stated, AI refers to intelligence demonstrated by machines versus natural intelligence displayed by humans or animals (Russell and Norvig 2010). In computer science, AI is defined as studying the design of intelligent agents (Poole et al. 1998). These intelligent agents refer to software and hardware components that perform tasks, which usually require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages (Crayton 2019).

Especially due to several breakthrough advancements made over the past years, AI has now been absorbed into almost every business sector, such as healthcare, retail, and automotive. In the coming years, AI's relatively fast growth is expected to gain momentum, and its prominent impact on business and society will also manifest. The recent advancements and the rising public awareness regarding AI technology can be attributed to the convergence of two factors. The first factor is the increasing availability of sufficient computing power, which is requisite for building high-functioning AI systems. Hardware manufacturers have particularly started to develop components that are specifically designed for AI applications, for example, Google's Tensor Processing Units¹² and Intel's Neural Network Processor¹³. These purported *application-specific integrated circuits* not only increase the computational power for innovative AI applications, but also help lower their costs via their higher efficiency. The second factor that fuels advancements in the AI field is the growing amount of data collected every day as a direct result of the digitalization of our society. On the one hand, analyzing and utilizing these masses of data have created a high demand for developing new AI-enabled autonomous systems and also attracting significant investments (CB Insights 2016). On the other hand, the large quantity of data also accelerates the development of more accurate AI systems, as data are a requisite input for training these systems. The more data an AI system are provided with, the better the AI system can learn its intended behavioral patterns. In

¹²<https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>

¹³<https://www.intel.ai/nervana-nnp/>

combination with sufficient computing power, the availability of training data also provides the necessary means to practitioners and researchers for designing better AI algorithms, which, in turn, creates new emerging fields of application for AI technology, as will be shown in the remainder of this chapter.

Artificial Intelligence

Artificial Intelligence refers to intelligence demonstrated by machines versus natural intelligence displayed by humans or animals (Russell and Norvig 2010).

The AI field draws upon a number of disciplines that contributed ideas, viewpoints, and techniques to AI. These disciplines include philosophy, mathematics, economics, neuroscience, psychology, computer engineering, cybernetics, and linguistics. Based on this, AI is able to tackle a number of different problems, such as reasoning, problem-solving, knowledge representation, planning, learning, natural language processing, perception, as well as the ability to move and manipulate objects (Russell and Norvig 2010). AI research has a twofold purpose: From a scientific point of view, AI aims at understanding the principles that make intelligent behavior possible in natural or artificial systems. From an engineering point of view, AI aims at designing and synthesizing useful, intelligent artifacts, for example, agents that act intelligently (Poole and Mackworth 2010). People have long been fascinated by the idea of using machines to perform complicated tasks. AI systems are colloquially known as machines that imitate cognitive functions, which humans associate with other human minds, such as learning and problem-solving (Russell and Norvig 2010). However, not all systems that are regarded as possessing AI fulfill these criteria to the same extent. Consequently, the AI field distinguishes between two different types of AI, namely strong and weak AI. Strong AI, also known as full AI or artificial general intelligence (AGI), is defined as “the intelligence of a machine that could successfully perform any intellectual task that a human being can” (Ryan 2014). For this, strong AI requires the ability to autonomously solve problems, plan, learn, adapt, and communicate in natural language, represent knowledge, and make reasonable judgments under uncertainty (Russell and Norvig 2010; Luger 2009; Poole et al. 1998). However, while much research effort is devoted to the implementation of AGI (AIG 2019), there are currently no examples of working systems that can claim all the above criteria and that can, therefore, be considered as strong AI. Existing AI systems are, therefore, all classified as weak AIs. Weak AI, which is occasionally also referred to as applied AI or artificial narrow intelligence (ANI), is “the use of software to study or accomplish specific problem solving or reasoning tasks” and is, thus, focused on one narrow task (Ryan 2014). A good example of weak AI systems is the current generation of consumer virtual assistants, such as Siri and Google Assistant, which were described in section 12.3. The functionality of these virtual assistants is limited to a pre-defined set of operations. The implemented AI is neither self-aware nor capable of developing new abilities by themselves to

solve new problems. In this sense, Siri and the other above mentioned consumer virtual assistants are not considered to have genuine intelligence. Beyond the distinction of strong and weak AI, future AI systems will be able to reason, plan, and solve problems autonomously for tasks, which they were not originally built for. These future AI systems, known as artificial super intelligence (ASI), will be self-aware and conscious systems that could be applied to any area (Kaplan and Haenlein 2019).

AI can also be considered by focusing on the business use of different types of AI systems. First, analytical AI systems have competencies that are related to, for example, pattern recognition and systematic thinking, i.e. cognitive intelligence. Examples of analytical AI systems include systems that are used for fraud detection in financial services, image recognition, and self-driving cars. Second, human-inspired AI systems not only have elements from cognitive intelligence, but also from emotional intelligence, such as adaptability, self-confidence, emotional self-awareness, and achievement orientation. In this sense, these systems can recognize human emotions, like joy, surprise, or anger, and can, therefore, be used during customer interactions or employee recruitments. Third, over and above cognitive and emotional intelligence, humanized AI systems have elements of social intelligence, such as empathy, teamwork, and inspirational leadership. These systems, using these elements, are able to be self-conscious and self-aware. Even though progress has been made in recognizing and mimicking human activities, humanized AI systems are still in the distant future (Kaplan and Haenlein 2019). An important technology that drives current advancements in the AI field is artificial neural networks, which simulate the functioning of the human brain in order to solve AI problems. These problems specifically include recognizing patterns in unstructured or highly complex datasets (Mthunzi et al. 2019). For this, a neural network has inputs, i.e. for datasets, outputs, i.e., for calculated results, and passages between them through various layers of simulated neural connections (see Fig. 12.6). These layers contain several processing nodes, i.e. artificial neurons, which are connected to other processing nodes in other layers via multiple input and output connections. A neural network is used by providing a dataset to the processing nodes of the network's input layer where it is split into individual values (Mthunzi et al. 2019). When a value is passed to a node through one or more of its input connections, the node applies a multiplier that is distinct for each input connected to the value. The node then adds these weighted values together. When the resulting value passes a node-specific threshold, the node sends the accumulated value to its successor nodes in the next layer. The values finally arriving at the output layer form the result calculated by the network. Changing the multipliers and threshold of individual nodes allows changing the path that values take and the manner in which they accumulated and, thus, change the output or behavior of the neural network. The process of configuring a neural network's behavior by adjusting node properties is called training. For this, the network is fed with training datasets for which the desired outcome is already known. By gradually adjusting the processing nodes, the network adapts its behavior for the given input until the desired output is generated, such as correctly recognizing objects in images or a robot's decision to take a certain action (Mthunzi et al. 2019).

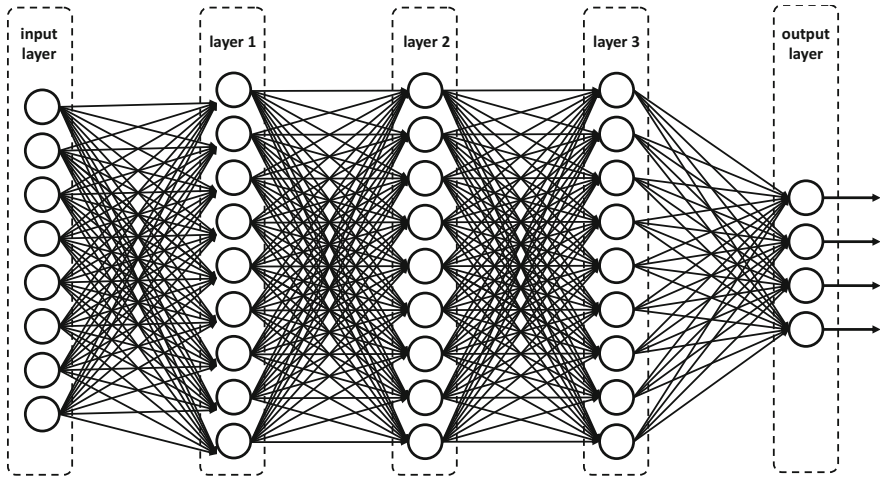


Fig. 12.6 Artificial Neural Network

Adding to this, adapting to new circumstances and detecting and extrapolating patterns are known as machine learning (Russell and Norvig 2010). Learning, as the word is used in this definition, generally refers to the agent's ability to improve its behavior based on experience (Poole and Mackworth 2010). This, in turn, brings about the distinction between two important terms, namely supervised and unsupervised learning. In supervised learning, the agent observes certain example input–output pairs and learns a function that maps from input to output. In unsupervised learning, however, the agent learns patterns in the input, even though no explicit feedback is supplied, for example, detecting potentially useful clusters of input examples (Russell and Norvig 2010).

Machine Learning

Machine learning is the science of making computers learn and act like humans and making computers improve their learning over time autonomously by feeding them data and information in the form of observations and real-world interactions (Mthunzi et al. 2019).

When considering the capabilities of this emerging technology called AI, its performance of a particular task is often compared to what humans are able to achieve. Nowadays, AI agents are capable of surpassing human-level competency in more complex tasks. For example, AI agents currently learn how to play what is considered the most complex board game of all times: Go (Saito et al. 2018).

AlphaGo, a computer program that Google developed to play the strategy board game Go, is a good example of radical novelty enabled by AI technologies like neural networks. Despite its relatively simple rules, Go is much more complex than

chess. Chess programs were able to play on par with human chess masters since the early 1980s (Hyatt and Nelson 1990). Owing to the larger board size, Go players have many more alternatives to consider for each move. While in chess, there are on average 400 different options for the next move, a Go move has about 130,000 options (Metzl 2019). Due to the resulting immense number of potential moves, traditional computer programs were not able to challenge experienced human players until recently. Nonetheless, with the help of a highly trained artificial neural network, AlphaGo AI managed to beat professional Go players - an achievement that was believed to be virtually impossible, considering the game's high complexity and dependence on human intuition and focused practice (The Japan Times 2017). The neural network of AlphaGo and its successors (i.e., AlphaGo Master, AlphaGo Zero, and AlphaZero) is trained solely with self-play reinforcement learning. This means that AlphaGo does not rely on human-generated data or supervision. The objective in this regard is that AlphaGo in itself predicts its move selections. The training via this artificial learning process leads to stronger self-play results and adapts with every training cycle of the underlying neural network (Saito et al. 2018; Dutta 2018).

Chronic disease management is a further area that requires recording longitudinal patient data (Sunyaev and Chornyi 2012). The computers' capability to record these massive amounts of information about patients currently transforms the healthcare domain. However, the enormous volume of data being collected, is impossible for human beings to analyze. Machine learning provides a suitable approach to automatically find patterns and make sense of the data, which enable physicians to deliver personalized healthcare services. According to an estimation of the World Health Organization (WHO), 422 million people worldwide have diabetes – with an upward tendency. This means that one out of eleven persons has to manage the chronic condition of this disease on a daily basis (The Medical Futurist 2018a). To address these issues, the Dublin-based medical technology company, Medtronic¹⁴, developed a smart, algorithm-based continuous glucose monitoring (CGM) system. CGM is specifically designed to support people who live with diabetes and who require daily multiple insulin injections. The system's main purpose is to predict a patient's glucose levels up to one hour in advance and, thus, allow for better diabetes management. Consequently, the CGM system uses a sensor that is worn by the diabetes patient. The sensor is connected to a smartphone to which it continuously transmits the patient's blood sugar levels. The accompanying smartphone app analyzes these data points with the help of an external AI system powered by IBM's AI platform, Watson Health¹⁵. Using AI algorithms, the system seeks to discover personal patterns in the patient's glucose level, such as changes in response to food intake, insulin dosages, physical activities, or other daily routines. Based on these patterns, the CGM system derives predictions that are used to notify patients about potential high and low glucose events up to 60 minutes before they actually

¹⁴<https://www.medtronic.com>

¹⁵<https://www.ibm.com/watson>

occur. This allows a patient to take countermeasures and keep her/his glucose level in its target range (Comstock 2018; Medtronic 2018).

Another contemporary example of the emergence of AI can be found in the automotive industry in the form of autonomous driving vehicles. These vehicles are capable of sensing their environment and moving with little or no human interventions (Gehrig and Stein 1999). While the first automated cars were developed in the 1970s, none have been brought to market yet (Weber 2014). With the help of new machine learning approaches, this is expected to change in the near future (Tian et al. 2018). One of these approaches is driven forward by the *Tesla Autopilot*, a driver-assistance system that is installed in electric cars produced by Tesla Inc.¹⁶ The system provides features like adaptive cruise control, lane centering, self-parking, automatic lane changing, and the ability to be summoned to and from a parking spot (Tesla 2019). To accomplish this, the Tesla Autopilot relies on cameras, ultrasonic sensors, radar, and onboard computing hardware that allow it to process sensor data using machine learning approaches. Besides onboard processing, Tesla cars permanently transmit data, which result from driving and using the Autopilot feature, over the Internet back to Tesla where it is used to train and further improve the underlying algorithms (Fehrenbacher 2015). The current vehicle generation, thereby, forms a network in which cars learn from each other and each individual driver essentially contributes to the training of an AI that will enable future car generations to drive fully autonomously. However, despite these prospects, autonomous driving vehicles are currently still associated with considerable uncertainties in terms of technical viability, traffic safety, and open legal questions, such as liability (Schreurs and Steuwer 2015; Beiker 2012; Awad et al. 2018), which contribute to the prevailing notion that AI in the automotive industry is still an emerging technology. In conclusion, this and the preceding chapters also show the great potential that lies in AI, particularly regarding its major role in encouraging the development of other emerging technologies, such as virtual assistants.

Summary

This chapter introduced the concept of emerging technology from a conceptual and an applicational perspective and, thereby, gave an overview of the emerging technologies' nature and their important implications. Conceptually, emerging technologies are defined by five key attributes, namely (1) radical novelty, (2) relatively fast growth, (3) coherence, (4) prominent impact, and (5) uncertainty and ambiguity. The five attributes capture the essence of the emerging technology concept and, thus, allow for differentiating between emerging and established technologies. The strengths of the attribute values are separate indicators for emergence, i.e. the

¹⁶<https://www.tesla.com>

attribute values change individually over time with the technology's progressing maturity. The key attributes can, therefore, also be used to discern three stages of emergence: (1) pre-emergence, (2) emergence, and (3) post-emergence. The pre-emergence stage (i.e., the earliest stage of emergence) is marked by high novelty, uncertainty, and ambiguity, as well as slow growth rates, low coherence, and little impact. During the emergence stage, the technology's growth, coherence, and impact start to increase, while novelty, uncertainty, and ambiguity begin to lessen. Finally, during the post-emergence stage (i.e., the late phase of emergence), the technology progresses toward becoming established and the five key attributes stabilize at either a high (growth, coherence, and impact) or low level (novelty, uncertainty, and ambiguity).

Regarding the applicational perspective, the first emerging technology discussed in this chapter was immersive technologies. Immersive technologies generate computer-based simulations of reality with physical, spatial, and visual dimensions that create a sense of immersion and that enhance the virtual experiences' realism. The concept incorporates different technological approaches of which two, VR and AR, were discussed in detail in this chapter. VR creates a fully enclosed non-physical world, which is intended to be experienced as being physically present through, for instance, using HMDs. In contrast, AR refers to virtual 3D objects in immersive reality, using digital devices, such as a smartphone. Both VR and AR technology exhibit high growth rates, stimulate the creation of radical novel applications, and become increasingly coherent. However, it is too early to label them as emerged, since the two technologies are still associated with high levels of uncertainty in terms of prevailing technical issues, user acceptance, appropriate incentivization, and understanding the long-term psychological effects.

Virtual assistant was the second emerging technology explained in this chapter. A virtual assistant can be classified as a novel type of personal software agent, acting on behalf of a human user in a purely computational environment. Virtual assistants utilize recent advances made in the field of AI to interpret human speech, execute the interpreted commands autonomously, and respond via synthesized voices. A virtual assistant variant that is expected to have a prominent impact is chatbots, particularly in the healthcare context. Chatbots are software agents that communicate interactively with human users via text or voice. In the healthcare context, it is expected that chatbots will make many consultations with physicians obsolete by directly addressing diagnosable health concerns. Furthermore, chatbots will support patients in managing their personal healthcare, thereby enabling patients to take more responsibility for their health. As an emerging technology, the virtual assistants' potential prominent impact is also accompanied by uncertainty and ambiguity. In the case of chatbots for healthcare applications, the viability of AI-based medical advice is still questioned, which forces patients to verify the virtual assistants' diagnoses with physicians.

Lastly, the emergence of AI has been detailed in the previous section. AI is defined as studying the design of intelligent agents, which comprise software and hardware components that autonomously perform tasks, which usually require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages. Owing to recent technical advancements, AI is

absorbed into almost every business sector, such as healthcare, retail, and automotive. An important technology that is driving current advancements in the AI field is artificial neural networks, which simulate the functioning of the human brain for the purpose of solving AI problems. These problems particularly include recognizing patterns in unstructured or highly complex datasets. To configure a neural network's behavior for a specific task, the network is provided with training datasets. The more data are used in this process, the more accurate is the network's output. Concluding this chapter, three examples of the emergence of AI in different contexts were described: AlphaGo (i.e., a computer program that plays the strategy board game Go), the continuous glucose monitoring (CGM) system (i.e., a system predicting a patient's glucose levels up to one hour in advance), and the Tesla Autopilot (i.e., an autonomous driver-assistance system for vehicles).

Questions

1. What attributes distinguish established technologies from emerging technologies?
2. What are the different stages an emerging technology goes through?
3. What are the differences between virtual reality (VR), augmented reality (AR), and mixed reality (MR)?
4. What are the reasons for the prominent impact of immersive technologies in the healthcare context?
5. What are the VR's and AR's unintended consequences for users?
6. What is a virtual assistant?
7. What are the reasons for the chatbots' prominent impact in the healthcare context?
8. What are the differences between strong AI and weak AI?
9. How do artificial neural networks work?
10. Why is artificial intelligence an emerging technology?

References

- Ada (2019) About ada. <https://ada.com/about/>. Accessed 18 Sept 2019
- Adner R, Levinthal D (2002) The emergence of emerging technologies. *Calif Manag Rev* 45 (1):50–66
- AIG (2019) Conference series on artificial general intelligence. <http://agi-conference.org/>. Accessed 18 Sept 2019
- Alexander J, Chase J, Newman N, Porter A, Roessner J (2012) Emergence as a conceptual framework for understanding scientific and technological progress. Paper presented at the PICMET'12: technology management for emerging Technologies, Vancouver, BC, 29 July–2 Aug 2012
- Althoff T, White RW, Horvitz E (2016) Influence of Pokémon go on physical activity: study and implications. *J Med Internet Res* 18(12):e315
- Arthur WB (2007) The structure of invention. *Res Policy* 36:274–287

- Awad E, Levine S, Kleiman-Weiner M, Dsouza S, Tenenbaum JB, Shariff A, Bonnefon J-F, Rahwan I (2018) Blaming humans in autonomous vehicle accidents: shared responsibility across levels of automation. <https://arxiv.org/abs/1803.07170>. Accessed 23 Sept 2019
- Azuma R (1997) A survey of augmented reality. *Presence Teleop Virt Environ* 6(4):355–385
- Azuma R, Bailiot Y, Behringer R, Feiner S, Julier S, MacIntyre B (2001) Recent advances in augmented reality. *IEEE Comput Graph Appl* 21(6):34–47
- Babylon (2019) Mission. <https://www.babylonhealth.com/about>. Accessed 18 Sept 2019
- Beiker SA (2012) Legal aspects of autonomous driving. *Santa Clara Law Rev* 52(4):1145–1156
- Boon W, Moors E (2008) Exploring emerging technologies using metaphors: a study of orphan drugs and pharmacogenomics. *Soc Sci Med* 66(9):1915–1927
- Botella C, Banos RM, Perpina C, Villa H, Alcaniz M, Rey A (1998) Virtual reality treatment of claustrophobia: a case report. *Behav Res Ther* 36:239–246
- Brown E, Cairns P (2004) A grounded investigation of game immersion. Paper presented at the CHI '04: extended abstracts on human factors in computing systems, Vienna, 24–29 Apr 2004
- Carmigniani J, Furht B (2011) Augmented reality: an overview. In: Furht B (ed) *Handbook of augmented reality*. Springer, New York, NY, pp 3–46
- CB Insights (2016) Artificial intelligence explodes: new deal activity record for AI startups. <https://www.cbinsights.com/research/artificial-intelligence-funding-trends/>. Accessed 18 Sept 2019
- Chao C-n, Zhao S (2013) Emergence of movie stream challenges traditional DVD movie rental—an empirical study with a user focus. *Int J Bus Admin* 4(3):22–29
- Chung H, Iorga M, Voas J, Lee S (2017) Alexa, can I trust you? *Computer* 50(9):100–104
- Comstock J (2018) FDA clears medtronic's smartphone-connected CGM for MDI users. <https://www.mobihealthnews.com/content/fda-clears-medtronics-smartphone-connected-cgm-mdi-users>. Accessed 18 Sept 2019
- Corrocher N, Malerba F, Montobbio F (2003) The emergence of new technologies in the ICT field: main actors, geographical distribution and knowledge sources. https://econpapers.repec.org/scripts/redir.pf?u=https%3A%2F%2Fwww.eco.uninsubria.it%2FRePEc%2Fpdf%2FQF2003_37.pdf;h=repec:ins:quaeco:qf0317. Accessed 18 Sept 2019
- Cozzens SE, Gatchair S, Kang J, Kim KS, Lee HJ, Ordóñez G, Porter A (2010) Emerging technologies: quantitative identification and measurement. *Technol Anal Strateg Manag* 22(3):361–376
- Crayton ED (2019) Redefining life sciences with artificial intelligence and blockchain. Ellen Debra Crayton
- Daponte P, De Vito L, Picariello F, Riccio M (2014) State of the art and future developments of the augmented reality for measurement applications. *Measurement* 57:53–70
- Day GS, Schoemaker PJH (2000) Avoiding the pitfalls of emerging technologies. *Calif Manag Rev* 42(2):8–33
- de Regt A, Barnes SJ (2019) V-commerce in retail: nature and potential impact. In: *Augmented reality and virtual reality*. Springer, Cham, pp 17–25
- Devezas TC (2005) Evolutionary theory of technological change: state-of-the-art and new approaches. *Technol Forecast Soc Chang* 72(9):1137–1152
- Dünnebeil S, Sunyaev A, Blohm I, Leimeister JM, Krcmar H (2012) Determinants of physicians' technology acceptance for e-health in ambulatory care. *Int J Med Inform* 81:746–760
- Dutta S (2018) Reinforcement learning with TensorFlow: a beginner's guide to designing self-learning systems with TensorFlow and OpenAI gym. Packt Publishing, Birmingham
- Dyer E, Swartzlander BJ, Gugliucci MR (2018) Using virtual reality in medical education to teach empathy. *J Med Libr Assoc* 106(4):498–500
- Eisenmann T, Lauren B, Liz K (2014) Google glass. *Harv Bus Sch Case Collect* 814-116:1–25
- Fehrenbacher K (2015) How Tesla is ushering in the age of the learning car. <http://fortune.com/2015/10/16/how-tesla-autopilot-learns/>. Accessed 18 Sept 2019
- Fenn J, Bloesch M (2018) Understanding Gartner's hype cycles. <https://www.gartner.com/en/documents/3887767>. Accessed 18 Sept 2019
- Freina L, Canessa A (2015) Immersive vs desktop virtual reality in game based learning. Paper presented at the 9th European conference on games based learning, Steinkjer, 8–9 Oct 2015

- Fryer LK, Nakao K, Thompson A (2019) Chatbot learning partners: connecting learning experiences, interest and competence. *Comput Hum Behav* 93:279–289
- Gaines J (2016) How do you make a young doctor really understand what it's like being 74? Virtual reality. *Upworthy*, 27 May 2016
- Gao F, Sunyaev A (2019) Context matters: a review of the determinant factors in the decision to adopt cloud computing in healthcare. *Int J Inf Manag* 48:120–138
- Gartner (2017) Gartner hype cycle. <https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/>. Accessed 19 Sept 2019
- Gartner (2018) Hype cycle for emerging technologies, 2018. <https://www.gartner.com/document/3885468>. Accessed 18 Sept 2019
- Gehrig SK, Stein FJ (1999) Dead reckoning and cartography using stereo vision for an autonomous car. Paper presented at the IEEE/RSJ international conference on intelligent robots and systems, Kyongju, 17–21 Oct 1999
- Groen EL, Bos JE (2008) Simulator sickness depends on frequency of the simulator motion mismatch: an observation. *Presence Virtual Augment Real* 17(6):584–593
- Halaweh M (2013) Emerging technology: what is it? *J Technol Manag Innov* 8(3):19–20
- Handa M, Aul G, Bajaj S (2012) Immersive technology – uses, challenges and opportunities. *Int J Comput Bus Res* 6(2):1–11
- Hettinger LJ, Riccio GE (1991) Visually induced motion sickness in virtual environments. *Presence Virtual Augment Real* 1(3):306–310
- Hino K, Asami Y, Lee JS (2019) Step counts of middle-aged and elderly adults for 10 months before and after the release of Pokémon GO in Yokohama, Japan. *J Med Internet Res* 21(2): e10724
- Hoy MB (2018) Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Med Ref Serv Q* 37(1):81–88
- Hung SC, Chu YY (2006) Stimulating new industries from emerging technologies: challenges for the public sector. *Technovation* 26(1):104–110
- Hyatt RM, Nelson HL (1990) Chess and supercomputers: details about optimizing cray blitz. Paper presented at the ACM/IEEE conference on supercomputing, New York, NY, 12–16 Nov 1990
- IKEA (2017) Try before you buy. New IKEA place app makes home furnishing easier. <https://www.ikea.com/gb/en/this-is-ikea/ikea-highlights/try-before-you-buy/>. Accessed 18 Sept 2019
- Janarthanam S (2017) Hands-on chatbots and conversational UI development. Packt, Birmingham
- Jennett C, Coxa AL, Cairns P, Dhoparee S, Epps A, Tijs T, Walton A (2008) Measuring and defining the experience of immersion in games. *Int J Hum Comput Stud* 66:641–661
- Joiner IA (2018) *Emerging library technologies: it's not just for geeks*, 1st edn. Elsevier, Cambridge, MA
- Kaplan A, Haenlein M (2019) Siri, Siri, in my hand: who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Bus Horiz* 62:15–25
- Kolasinski EM (1995) Simulator sickness in virtual environments. <https://apps.dtic.mil/dtic/tr/fulltext/u2/a295861.pdf>. Accessed 18 Sept 2019
- KPMG International (2018) The pulse of fintech 2018: biannual global analysis of investment in fintech. <https://home.kpmg/content/dam/kpmg/us/pdf/2018/07/pof-1H-18-report.pdf>. Accessed 18 Sept 2019
- Kudina O, Verbeek P-P (2018) Ethics from within: google glass, the collingridge dilemma, and the mediated value of privacy. *Sci Technol Hum Values* 44(2):291–314
- Leydesdorff L, Cozzens S, van den Besselaar P (1994) Tracking areas of strategic importance using scientometric journal mappings. *Res Policy* 23(2):217–229
- Lopatovska I, Rink K, Knight I, Raines K, Cosenza K, Williams H, Sorsche P, Hirsch D, Li Q, Martinez A (2018) Talk to me: exploring user interactions with the Amazon Alexa. *J Librariansh Inf Sci* 51(4):984–997
- Luciani A (2007) Virtual reality and virtual environment. In: Luciani A, Cadoz C (eds) *Enaction and enactive interfaces: a handbook of terms*. Enactive Systems Books, pp 299–300
- Luger GF (2009) *Artificial intelligence: structures and strategies for complex problem solving*, 6th edn. Pearson Education, Boston, MA

- Mandl KD, Kohane IS (2017) 21st-century health IT system – creating a realworld information economy. *N Engl J Med* 376(20):1905–1907
- Martin BR (1995) Foresight in science and technology. *Tech Anal Strat Manag* 7(2):139–168
- Mazloumi GA, Walker FR, Hodgson DM, Nalivaiko E (2018) A comparative study of cybersickness during exposure to virtual reality and “classic” motion sickness: are they different? *J Appl Physiol* 125(6):1670–1680
- Medtronic (2018) Medtronic receives FDA approval for guardian connect continuous glucose monitoring (CGM) system for people living with diabetes. <https://www.globenewswire.com/news-release/2018/03/12/1420694/0/en/Medtronic-Receives-FDA-Approval-for-Guardian-Connect-Continuous-Glucose-Monitoring-CGM-System-for-People-Living-with-Diabetes.html>. Accessed 18 Sept 2019
- Merriam-Webster (2019a) Definition of augmented reality. <https://www.merriam-webster.com/dictionary/augmented%20reality>. Accessed 19 Sept 2019
- Merriam-Webster (2019b) Merriam-Webster’s collegiate dictionary. <https://www.merriam-webster.com/>. Accessed 19 Sept 2019
- Metzl J (2019) Hacking Darwin: genetic engineering and the future of humanity. Sourcebooks, Naperville, IL
- Milgram P, Kishino F (1994) A taxonomy of mixed reality visual displays. *IEICE Trans Inf Syst* E77-D(12):1321–1329
- Mitchell SD (2007) The import of uncertainty. *Pluralist* 2(1):58–71
- Mthunzi SN, Benkhelifa E, Bosakowski T, Hariri S (2019) A bio-inspired approach to cyber security. In: Gupta BB, Sheng M (eds) *Machine learning for computer and cyber security: principle, algorithms, and practices*. CRC Press, Boca Raton, FL, pp 75–105
- Ortiz de Gortari AB, Griffiths MD (2015) Game transfer phenomena and its associated factors: an exploratory empirical online survey study. *Comput Hum Behav* 51:195–202
- Parvinen P, Hamari J, Pöyry E (2019) Introduction to minitrack: mixed, augmented and virtual reality: co-designed services and applications. Paper presented at the 52nd Hawaii international conference on system sciences, Maui, HI, 8–11 Jan 2019
- Pillay R (2018) *Healthcare 3.0: how technology is driving the transition to prosumers, platforms and outsourcing*. Xlibris US, Bloomington, IN
- Poole DL, Mackworth AK (2010) *Artificial intelligence. Foundations of computational agents*. Cambridge University Press, New York, NY
- Poole D, Mackworth AK, Goebel R (1998) *Computational intelligence: a logical approach*. Oxford University Press, New York, NY
- Porter AL, Ashton WB, Clar G, Coates JF, Cuhls K, Cunningham SW, Ducatel K, van der Duin P, Georgehiou L, Gordon T, Linstone H, Marchau V, Massari G, Miles I, Mogee M, Salo A, Scapolo F, Thissen W (2004) *Technology futures analysis: toward integration of the field and new methods*. *Technol Forecast Soc Chang* 71(3):287–303
- Press CU (2019) Cambridge dictionaries online. <https://dictionary.cambridge.org/>. Accessed 19 Sept 2019
- Pulse (2017) Babylon GP appointment app raises fears of patients ‘gaming’ system. <http://www.pulsetoday.co.uk/your-practice/practice-topics/it/babylon-pilot-ditched-after-patients-manipulated-system-to-book-gp-appointments/20035712.article>. Accessed 18 Sept 2019
- Raskar R, Low K (2001) Interacting with spatially augmented reality. Paper presented at the 1st international conference on computer graphics, virtual reality and visualisation, Camps Bay, Cape Town, 5–7 Nov 2001
- Rauschnabel PA, He J, Ro YK (2018) Antecedents to the adoption of augmented reality smart glasses: a closer look at privacy risks. *J Bus Res* 92(1):374–384
- Reardon S (2014) Text-mining offers clues to success. *Nature* 509(7501):410
- Rotolo D, Hicks D, Martin BR (2015) What is an emerging technology? *Res Policy* 44(10):1827–1843
- Ruddle RA (2004) The effect of environment characteristics and user interaction on levels of virtual environment sickness. Paper presented at the IEEE virtual reality, Chicago, IL, 27–31 Mar 2004
- Russell SJ, Norvig P (2010) *Artificial intelligence. A modern approach*, 3rd edn. Pearson Education, Upper Saddle River, NJ

- Ryan M (2014) *The digital mind: an exploration of artificial intelligence*. CreateSpace Independent Publishing Platform, North Charleston, SC
- Saito S, Wenzhuo Y, Shanmugamani R (2018) *Python reinforcement learning projects: eight hands-on projects exploring reinforcement learning algorithms using TensorFlow*. Packt Publishing, Birmingham
- Schreurs M, Steuwer S (2015) Autonomous driving – political, legal, social, and sustainability dimensions. In: Maurer M, Gerdes J, Lenz B, Winner H (eds) *Autonomes Fahren*. Springer, Berlin, pp 151–173
- Sciuto A, Saini A, Forlizzi JL, Hong JI (2018) “Hey Alexa, what’s up?”: a mixed-methods studies of in-home conversational agent usage. Paper presented at the designing interactive systems conference, Hong Kong, 9–13 June 2018
- Small World (2017) Breastfeeding support program through google glass. <https://www.smallworldsocial.com/breastfeeding-support-project/>. Accessed 18 Sept 2019
- Small H, Boyack KW, Klavans R (2014) Identifying emerging topics in science and technology. *Res Policy* 48(8):1450–1467
- Soares C, Simão E (2019) Immersive multimedia in information revolution. In: Simao E, Soares C (eds) *Trends, experiences, and perspectives in immersive multimedia and augmented reality. Advances in multimedia and interactive technologies*. IGI Global, Hershey, PA, pp 192–210
- Sommerauer P, Müller O (2018) Augmented reality in informal learning environments: investigating shortterm and long-term effects. Paper presented at the 51st Hawaii international conference on system sciences, Hilton Waikoloa Village, HI, 3–6 Jan 2018
- Stahl BC (2011) What does the future hold? A critical view on emerging information and communication technologies and their social consequences. In: Chiasson M, Henfridsson O, Karsten H, DeGross JI (eds) *Researching the future in information systems: IFIP WG 8.2 working conference, Future IS 2011, Turku, 6–8 June 2011*. Springer, Heidelberg, pp 59–76
- Steuer J (1992) Defining virtual reality: dimensions determining telepresence. *J Commun* 42(4):73–93
- Stevenson A, Lindberg CA (2010) *New Oxford American dictionary*, 3rd edn. Oxford University Press, Oxford
- Stirling A (2007) Risk, precaution and science: towards a more constructive policy debate. *EMBO Rep* 8(4):309–315
- Suh A, Prophet J (2018) The state of immersive technology research: a literature analysis. *Comput Hum Behav* 86:77–90
- Sunyaev A, Chorny D (2012) Supporting chronic disease care quality: design and implementation of a health service and its integration with electronic health records. *ACM J Data Inf Qual* 3(2):3–21
- Sutherland IE (1964) Sketchpad a man-machine graphical communication system. *Simulation* 2(5): R-3–R-20
- Tesla (2019) Autopilot: future of driving. <https://www.tesla.com/autopilot>. Accessed 18 Sept 2019
- The Guardian (2016) Cutting-edge theatre: world’s first virtual reality operation goes live. <https://www.theguardian.com/technology/2016/apr/14/cutting-edge-theatre-worlds-first-virtual-reality-operation-goes-live>. Accessed 18 Sept 2019
- The Japan Times (2017) AlphaGo AI stuns go community. <https://www.japantimes.co.jp/opinion/2017/06/03/editorials/alphago-ai-stuns-go-community/>. Accessed 18 Sept 2019
- The Medical Futurist (2016a) 5 ways medical virtual reality is already changing healthcare. <https://medicalfuturist.com/5-ways-medical-vr-is-changing-healthcare>. Accessed 18 Sept 2019
- The Medical Futurist (2016b) Augmented reality in healthcare will be revolutionary. <https://medicalfuturist.com/augmented-reality-in-healthcare-will-be-revolutionary>. Accessed 18 Sept 2019
- The Medical Futurist (2017a) The 10 most exciting digital health stories of 2017. <https://medicalfuturist.com/10-exciting-digital-health-stories-2017>. Accessed 18 Sept 2019
- The Medical Futurist (2017b) Chatbots will serve as health assistants. <https://medicalfuturist.com/chatbots-health-assistants>. Accessed 18 Sept 2019

- The Medical Futurist (2018a) Medtronic's smart continuous glucose monitoring system rolls out this summer. <https://medicalfuturist.com/medtronics-smart-continuous-glucose-monitoring-system-rolls-summer>. Accessed 18 Sept 2019
- The Medical Futurist (2018b) The top 12 health chatbots. <https://medicalfuturist.com/top-12-health-chatbots>. Accessed 18 Sept 2019
- The Peak (2016) SFU students design virtual reality game for cancer patients. <http://the-peak.ca/2016/05/sfu-students-design-virtual-reality-game-for-cancer-patients/>. Accessed 18 Sept 2019
- Tian Y, Pei K, Jana S, Ray B (2018) DeepTest: automated testing of deep-neural-network-driven autonomous cars. Paper presented at the 40th international conference on software engineering, Gothenburg, 27 May–3 June 2018
- Warman M (2013) Google glass: we'll all need etiquette lessons. The telegraph. <https://www.telegraph.co.uk/technology/google/10015697/Google-Glass-well-all-need-etiquette-lessons.html>. Accessed 18 Sept 2019
- Weber M (2014) Where to? A history of autonomous vehicles. <https://www.computerhistory.org/atc/m/where-to-a-history-of-autonomous-vehicles/>. Accessed 18 Sept 2019
- Witmer BG, Singer MJ (1998) Measuring presence in virtual environments: a presence questionnaire. *Presence Teleop Virt Environ* 7(3):225–240
- Zahedi FM, Walia N, Jain H (2016) Augmented virtual doctor office: theory-based design and assessment. *J Manag Inf Syst* 33(3):776–808
- Zion Market Research (2019) Healthcare chatbots market by deployment model (Cloud-based and on-premise), by component (services and software), by end-user (Healthcare providers, insurance companies, patients, and others), and by application (Medical guidance & appointment scheduling and medication assistance & symptom checking): global industry perspective, comprehensive analysis, and forecast, 2018–2025. <https://www.zionmarketresearch.com/report/healthcare-chatbots-market>. Accessed 18 Sept 2019

Further Reading

- Fryer LK, Nakao K, Thompson A (2019) Chatbot learning partners: connecting learning experiences, interest and competence. *Comput Hum Behav* 93:279–289
- Furht B (2011) Handbook of augmented reality. Springer, New York, NY
- Janarthanam S (2017) Hands-on chatbots and conversational UI development. Packt Publishing, Birmingham
- Luger GF (2009) Artificial intelligence: structures and strategies for complex problem solving, 6th edn. Pearson Education, Boston, MA
- Poole D, Mackworth AK, Goebel R (1998) Computational intelligence: a logical approach. Oxford University Press, New York, NY
- Rotolo D, Hicks D, Martin BR (2015) What is an emerging technology? *Res Policy* 44(10):1827–1843
- Russell SJ, Norvig P (2010) Artificial intelligence. A modern approach, 3rd edn. Pearson Education, Upper Saddle River, NJ
- Stahl BC (2011) What does the future hold? A critical view on emerging information and communication technologies and their social consequences. In: Chiasson M, Henfridsson O, Karsten H, DeGross JI (eds) Researching the future in information systems: IFIP WG 8.2 Working Conference, Future IS 2011, Turku, 6–8 June 2011. Springer, Heidelberg, pp 59–76
- Zahedi FM, Walia N, Jain H (2016) Augmented virtual doctor office: theory-based design and assessment. *J Manag Inf Syst* 33(3):776–808

Glossary

Application Service Provision Application service provision is a form of outsourcing in which firms rent packaged software and associated services from a third party.

Architectural Design (Outcome Perspective) Architectural design refers to the architectural design process's outcome, that is the collection of hardware and software components and their interfaces, which makes up the framework for the development of an information system.

Architectural Design (Process Perspective) Architectural design refers to the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of an information system.

Architectural Model An illustration, created using available standards, in which the primary concern is to represent the architecture of an IS from a specific perspective and for a specific purpose.

Architectural Pattern An architectural pattern is an abstract description of a recommended architectural approach that has been tested and proven in different information systems and environments.

Artificial Intelligence Artificial Intelligence refers to intelligence demonstrated by machines versus natural intelligence displayed by humans or animals.

Augmented Reality Augmented reality describes an enhanced version of reality that is created by using technology to overlay digital information on an image of a thing, which is viewed through a device, such as a smartphone camera.

Blockchain Blockchain is a Distributed Ledger Technology concept comprising a chain of cryptographically linked, chronologically ordered, 'blocks' containing batched transactions.

Certification Certification is defined as a third-party attestation of products, processes, systems, or persons that verifies conformity to specified criteria.

Chatbot Chatbots refer to software agents that are designated for interactive communication with humans via text or voice.

Cloud Auditors A cloud auditor is a party that performs an independent assessment to determine whether a cloud service fulfills given requirements.

Cloud Brokers A cloud broker is an entity that manages the use, performance, and delivery of cloud services, and that negotiates relationships between cloud providers and cloud customers.

Cloud Computing Cloud computing is a model that enables ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources that can rapidly be provisioned at any time and from any location via the Internet or a network.

Cloud Customer A cloud customer represents a person or organization that maintains a business relationship with, and uses the service a cloud provider offers.

Cloud Data Center Operators Cloud data center operators provide the basic IT infrastructure, including server rooms, redundant or backup components, infrastructure for power supply, data communications connections, environmental controls (e.g., cooling, air conditioning and fire emergency equipment), and various security devices.

Cloud Provider A cloud provider is a legal person or an organization that is responsible for making a cloud service available to interested parties.

Computer Network A computer network is a collection of computers and devices connected for the purpose of electronic data communication that allows them to share information and services.

Consensus Mechanism A consensus mechanism is designed to achieve agreement on the respective state of replications of stored data between a distributed database's nodes under consideration of network failures.

Content Delivery Network (CDN) Content Delivery Network (CDN) is a type of content network in which the content network elements are arranged for more effective delivery of content to clients. A CDN usually comprises a request-routing system, surrogates, a distribution system, and an accounting system.

Continuous Cloud Service Certification Continuous cloud service certification includes automated monitoring and auditing techniques, as well as mechanisms for transparent provision of certification-relevant information to continuously confirm adherence to certification criteria.

Critical Information Infrastructure An information system that, if it is disrupted or has unintended consequences, can have detrimental effects on vital societal functions or the health, safety, security, or economic and social well-being of people.

Critical Infrastructure A system that is essential for the maintenance of vital societal functions or the health, safety, or economic and social well-being of people.

Decoupling Decoupling describes a process in which one element of a system becomes an independent service with a defined service interface so that internal changes in this service will not disrupt the functioning of other dependent elements.

Distributed Database A distributed database is a type of database where data are replicated across multiple storage devices (nodes) with equal rights.

Distributed Ledger A distributed ledger is a type of distributed database that assumes the presence of nodes with malicious intentions. A distributed ledger comprises a ledger's multiple replications in which data can only be appended or read.

Distributed Ledger Technology Distributed Ledger Technology (DLT) enables the realization and operation of distributed ledgers, which allow benign nodes, through a shared consensus mechanism, to agree on an (almost) immutable record of transactions despite Byzantine failures and eventually achieving consistency.

Distributed System A distributed system is a collection of independent computers that appears to its users to be a single coherent system.

Distributed System Emerging Technology An emerging technology is a radically novel and relatively fast-growing technology characterized by a certain degree of coherence persisting over time and with the potential to exert a considerable impact on the socio-economic domain(s). This impact is observed in terms of the composition of actors, institutions, and patterns of interactions among these, along with the associated knowledge production processes. An emerging technology's most prominent impact, however, lies in the future and is, therefore, still somewhat uncertain and ambiguous in the emergence phase.

Domain Name System (DNS) The Domain Name System (DNS) is a hierarchically structured, distributed set of databases that maps IP addresses to corresponding domain names.

Edge Computing Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services.

Extensible Markup Language (XML) Extensible Markup Language, commonly abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of the computer programs which process them.

Fog Computing Fog computing is a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes (physical or virtual), residing between smart end-devices and centralized (cloud) services.

Fog Nodes The fog node is the core component of the fog computing architecture. Fog nodes are either physical components (e.g., gateways, switches, routers, servers) or virtual components (e.g., virtualized switches, virtual machines, cloudlets, etc.)

that are tightly coupled with the smart end-devices or access networks and provide computing resources to these devices.

Functional Requirement Functional requirements define the desired features and functions of a system or one of its components. A functional requirement includes the definition of a functionality and its transformation from an input into a desired output.

Grid Computing Grid Computing enables resource sharing and coordinated problem solving in dynamic, multi-institutional, and large-scale geographically-distributed virtual organizations.

Hyper Text Transfer Protocol (HTTP) The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.

Immersive Technologies Immersive technologies generate computer-based simulations of reality with physical, spatial, and visual dimensions that create a sense of immersion and enhancing the virtual experiences' realism.

Industrial Internet of Things The Industrial Internet of Things is the application of Internet of Things technologies (e.g., smart objects and smart devices) within an industrial setting in order to promote goals that are unique to industry. It covers diverse industries, such as energy, manufacturing, agriculture, health care, retail, logistics, and aviation.

Information Systems Information systems are interrelated components that work together to collect, process, store, and disseminate information to support decision-making, coordination, control, analysis, and visualization in an organization.

Information System Architecture Fundamental concepts or properties of an information system in its environment, as embodied in its elements and relationships, and in the principles of its design and evolution.

Internet Computing Internet computing is concerned with the applications provided on the Internet, the architectures and technologies used in applications on the Internet, and the systemic matters that shape the design of applications on the Internet. Internet computing encompasses all applications irrespective of whether they are built for the general public (e.g., social network services), or solely used in a single organization (e.g., enterprise resource planning systems) or in a closed group of organizations (e.g., supply chain management systems).

Internet of Things The Internet of Things is a self-configuring, adaptive, and complex network that interconnects "things" which have a physical and virtual representation on the Internet according to standard communication protocols. It entails nine essential characteristics: (1) interconnection of things; (2) connection of things to the Internet; (3) uniquely identifiable things; (4) ubiquity; (5) sensing (and actuation) capabilities; (6) embedded intelligence; (7) interoperable communication capabilities; (8) self-configurability; and (9) programmability.

Internet Protocol Suite The Internet protocol suite is a set of protocols that enables Internet communication by specifying data transmission, addressing, and routing.

IP Address An IP address is a group of binary numbers uniquely identifying a node within a network that uses the Internet Protocol.

IT Outsourcing IT outsourcing refers to transferring some or all of the IT related decision-making rights, business processes, internal activities, and services to external providers.

Machine Learning Machine learning is the science of making computers learn and act like humans and making computers improve their learning over time autonomously by feeding them data and information in the form of observations and real-world interactions.

Message-Oriented Middleware Message-oriented middleware is any middleware infrastructure that provides messaging capabilities. It provides a means to build distributed systems, where distributed processes communicate through messages exchanged via message queuing or message passing.

Middleware Middleware is a type of software used to manage and facilitate interactions between applications across computing platforms.

Mist Computing Mist computing is a lightweight and rudimentary form of fog computing that resides directly within the network fabric at the edge of the network fabric, bringing the fog computing layer closer to the smart end-devices. Mist computing uses microcomputers and microcontrollers to feed into fog computing nodes and potentially onward towards the centralized (cloud) computing services.

Network Function Virtualization Network Functions Virtualization (NFV) describes the architectural principle of separating network functions (i.e., functional blocks within a network infrastructure that have well-defined external interfaces and well-defined functional behavior) from the hardware on which they run by using virtual hardware abstraction.

Nonfunctional Requirements Nonfunctional requirements are requirements that are not specifically concerned with the system's functionality, but rather define general quality attributes and constraints.

Object-Oriented Middleware Object-Oriented Middleware is defined as a middleware infrastructure that offers object-oriented principles for the development of distributed systems.

Overlay Network An overlay network is a virtual network of nodes and logical links built on top of an existing network in order to implement a network service not available in the existing network.

Packet Switching Packet switching describes a switching and transmission technology which splits complete messages into smaller packets. These packets can be

transmitted via a network's different lines and the receiving host reassembles them into the original message.

Platformization Platformization characterizes the process by which an entity (a provider organization) creates access and interaction opportunities centered around a core bundle of services (the platform) within an ecosystem of customers, complementors (i.e., cloud service partners), and other stakeholders.

Recombination Recombination refers to the process in which innovation potential is generated by combining cloud services and integrating them with other transformative technologies within and across platform ecosystems.

Remote Procedure Call A remote procedure call is the synchronous language-level transfer of control between programs in disjoint address spaces whose primary communication medium is a narrow channel.

Router On a network, a router is a device that determines the best path for forwarding a data packet toward its destination. A router is connected to at least two networks and is located at the gateway where one network meets another.

Service A service is a logical representation of a repeatable business activity that has a specified outcome, is self-contained, may consist of other services, and is a black box to the service's consumers.

Service Oriented Architecture (SOA) Service oriented architecture represents a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functions that are built as software components, i.e. discrete pieces of code and/or data structures that can be reused for different purposes.

Smart Cities Smart cities are sustainably developed urban areas that offer a high quality of living. They outperform non-smart cities in the spheres of economy, people, governance, mobility, environment, and living through a combination of strong human and social capital, and a modern, pervasive ICT infrastructure.

Smart Device Smart devices are portable multi-purpose information and communication technology (ICT) devices that enable access to several application services located on the device or remotely on servers. They are usually owned and used by one person. Examples of smart devices include lap-tops, mobile phones, and tablets.

Smart Environment A smart environment is a physical world interwoven with a multitude of sensors, actuators, displays, and computing elements, seamlessly interacting with everyday objects of people's lives and connected via a continuous network.

Smart Factory A smart factory is a factory interspersed with sensors and smart objects that are connected to each other in a factory-spanning network. This network allows for sensing and tracking goods, as well as production flows, and provides additional context information. Smart factories have several advantages over

traditional factories and, for example, allow for optimizing production flows and inventory management.

Smart Homes Smart homes are a kind of smart environment. They are homes that are interspersed with smart objects and devices, which allow for homes to be aware of what happens inside. Smart homes strive to improve living comfort, while they simultaneously decrease resource usage and increase security.

Smart Object Smart objects are physical objects with an ability to recognize, process, and exchange data, act autonomously, adapt flexibly to specific situations, and interact with humans, as well as other objects and smart objects. Standard examples of smart objects include intelligent light bulbs, mugs, and heaters.

Software Defined Networking Software Defined Networking (SDN) is an emerging network architecture in which network control is decoupled from the forwarding devices and is directly programmable.

Transaction-Oriented Middleware Transaction-oriented middleware is any middleware infrastructure that supports the execution of electronic transactions in a distributed setting.

Virtual Assistant Virtual assistants refer to software agents that are able to interpret human speech, autonomously execute the user-issued command, and respond via synthesized voices.

Virtual Reality Virtual Reality refers to technology that generates an interactive virtual environment that is designed to simulate a real-life experience.

Web Service Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, and invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web-based.

World Wide Web The World Wide Web (WWW, or simply the Web) is an information space (on the Internet) in which global identifiers called Uniform Resource Identifiers identify the items of interest, referred to as resources.

XML Schema Definition (XSD) Schemata An XSD schema aims at defining and describing a class of XML documents via schema components to constrain and document the meaning, usage, and relationships of their constituent parts: datatypes, elements, as well as their content, attributes and values.